Machines that learn how to code open-ended survey data. Part I: the basic approach and a working system

Andrea Esuli, Tiziano Fagni, Fabrizio Sebastiani

2009-TR-011

Machines that Learn how to Code Open-Ended Survey Data. Part I: The Basic Approach and a Working System

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 the last seven years we have carried out experimental research aimed at developing software that automatically codes open-ended survey responses. These projects have led to the generation of an industrial-strength software package now in operation at the Customer Insight division of a large international banking group, and now integrated into a widely-used software platform for the management of open-ended survey data. This software, which can code data at a rate of tens of thousands of open-ended responses per hour, and that can address responses formulated in any of five major European languages, is the result of contributions from different fields of computer science, including Information Retrieval, Machine Learning, Computational Linguistics, and Opinion Mining. Our approach is based on a *learning* metaphor, whereby automated verbatim coders are *automatically* generated by a general-purpose process that learns, from a user-provided sample of manually coded verbatims, the characteristics that new, uncoded verbatims should have in order to be attributed the codes in the codeframe. In this paper we discuss the basic philosophy underlying this software. In a forthcoming companion paper we present the results of experiments we have run on several datasets of real respondent data in which we have compared the accuracy of the software against the accuracy of human coders.

Keywords: Survey coding, open-ended questions, open-ended responses, automatic coding, machine learning, opinion mining, sentiment analysis

1. Introduction

Coding verbatim responses is a bit like doing the dishes after hosting a dinner party: a somewhat tedious and time-consuming experience, but ultimately satisfying when you see the results stacked neatly away, ready for use later. At least, that was the case before dishwashers became commonplace.

[Tim Macer, Quirk's Marketing Research Review, 16(7), 2002.]

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

Open-ended questions are an important way to obtain informative data in surveys, and this is so for a variety of applications, including market research, customer relationship management, enterprise relationship management, and opinion research in the social and political sciences (Reja et al., 2003; Schuman and Presser, 1979). Closed questions generate data that are certainly more manageable, but suffer from several shortcomings, since they straightjacket the respondent into conveying her thoughts and opinions into categories that the questionnaire designer has developed *a priori*; as a result, a lot of information that the respondent might potentially provide is lost. On the contrary, open-ended questions allow freedom of thought, and the responses that are returned may provide perspectives and slants that had not been anticipated by the questionnaire designer, thus providing far richer information on the opinions of the respondent. Furthermore, asking an open question tells the respondent that her opinions are seriously taken into account and her needs are cared of; the same cannot be said of closed questions, since these may instead convey the impression that the interviewers are only interested in orthodox responses and orthodox respondents. For instance, an unhappy customer certainly knows that she can threaten to defect to the competition by answering an open question, but certainly knows that she will never be able to voice such a discontent through closed questions: no questionnaire designer would include a question such as "Do you intend to defect to the competition? (Yes/No)".

Unfortunately, the price one has to pay for including open-ended questions in a questionnaire is a much greater difficulty in using the data obtained by the respondents. Once the designer has developed a codeframe (aka "codebook") for the question under consideration, a human coder needs to read the returned answers one by one in order to assign them the appropriate codes; this may require a huge investment in humanpower, depending on the size of the respondents' pool, and does not lend itself to fast turnaround of results.

In the recent past, computer-assisted methods of coding open-ended verbatim responses (henceforth: "verbatims") have been proposed as a solution. Unfortunately, they still fall short of truly automating the coding process, since they all require a lot of human involvement in the coding process. Some of these systems, such as Confirmit[1] (O'Hara and Macer, 2005), Voxco's Command Center[2] (Macer, 2007b), SPSS' mrInterview[3], and Snap[4], are (as far as open-ended questions are concerned)

[1] http://www.confirmit.com/
[2] http://www.voxco.com/
[3] http://www.spss.com/mrinterview/
[4] http://www.snapsurveys.com/

essentially powerful, user-friendly systems that assist and enhance the productivity of the user in manually coding the verbatims: the only difference with coding as it was performed before computers were born, is that all the objects of interest are in digital form, so that no paper and pencil is involved. Some other systems such as Language Logic's Ascribe™[5] (Macer, 2002), SphinxSurvey[6] (Macer, 1999), streamBASE GmbH's Coding-Modul[7] (Macer, 2007a), SPSS' Text Analysis for Surveys[8], Provalis Research's Wordstat[9] (Macer, 2008), and the no longer available Verbastat (Macer, 2000), further assist the user by means of (sometimes sophisticated) word searching, text matching, or "text mining" capabilities; still, the decision whether a given code should or should not be attributed to a verbatim ultimately rests with the user. Other systems, such as iSquare's i2 SmartSearch[10], rely on the user to engineer rules for automated verbatim coding; while these rules are indeed capable of automating the coding process, the human effort involved in writing the rules is still high, and the level of expertise required for this is high as well.

In this paper we propose instead a radically different approach to developing automated verbatim coding systems. The approach is based on a *learning* metaphor, whereby automated verbatim coders are *automatically* generated by a general-purpose process that learns, from a user-provided sample of manually coded verbatims, the characteristics that new, uncoded verbatims should have in order to be attributed the codes in the codeframe. This approach adds a further level of automation to the methods described above, since no human involvement is required aside from that of providing a sample of manually coded verbatims. The net effect is that any human coder, and not necessarily a computer-savvy one, may set up and operate such a system in full autonomy.

In the rest of the paper we will exemplify this approach by describing an industrial-strength software system (dubbed VCS™, for V̲erbatim C̲oding S̲ystem) that we have developed along these lines. This system is now in operation at the Customer Insight division of a large international banking group, and is now available from within a widely-used software platform for the management of open-ended survey data. This software, which can code data at a rate of tens of thousands of open-ended verbatims per hour, and that can address responses formulated

[5] `http://www.languagelogic.info/`
[6] `http://www.sphinxsurvey.com/en/home/home_sphinx.php`
[7] `http://www.streambase.de/`
[8] `http://www.spss-sa.com/spss_text_analysis_for_surveys.html`
[9] `http://www.provalisresearch.com/wordstat/Wordstat.html`
[10] `http://www.isquare.de/i2SmartSearch.htm`

in any of five major European languages, is the result of the authors' basic research efforts in different fields of computer science, including Information Retrieval (see e.g., (Manning et al., 2008)), Machine Learning and Pattern Recognition (see e.g., (Duda et al., 2001; Hastie et al., 2001)), Computational Linguistics (see e.g., (Mitkov, 2003)), and Opinion Mining and Sentiment Analysis (see e.g., (Pang and Lee, 2008)).

The rest of this paper is organized as follows. Section 2 describes the basic philosophy underlying the machine learning approach to verbatim coding and the overall mode of operation of the VCS™ system. Section 3 adds further insight into VCS™ by giving, in question and answer format, a number of clarifications on the workings of VCS™. Section 4 looks at some other features that are available in VCS™ and at some other features that are going to be available in the next release, while Section 5 concludes.

2. VCS™, an automated verbatim coding system

2.1. A short history of VCS™

The very first ancestor of VCS™ was a fairly primitive system that the third author and a colleague developed and tested on three datasets from the 1996 General Social Survey (a continuous social survey that aims at investigating how people assess their physical and mental health, external and internal security threats, etc.) run by the US National Opinion Research Center (NORC); the results of these tests were published in (Giorgetti and Sebastiani, 2003). We will refer to this as version 0.1 of VCS™.

In 2006 the first licence for what had by then become an industrial-strength system (hereafter dubbed VCS 1.0) was issued to Egg plc[11] (the largest purely online banking group in the world, now part of Citigroup[12]), for use at their Customer Insight division. This version is still in operation at Egg, where it manages all of Egg's customer satisfaction verbatim data, with an estimated amount (as of 2006) of more than 20,000 verbatims per month, and with huge backlogs being now processed for retrospective data analysis[13]. VCS™ 1.0 operates as a Web-enabled application which interacts with the Confirmit Web

[11] http://www.egg.com/

[12] http://www.citigroup.com/

[13] VCS™ 1.0 has obtained considerable success in the market research community, winning the "2007 Best New Thinking Award" from the Market Research Society, the "2006 Amerigo Vespucci Award" for Market Research from Confindustria (the Italian Industralists Association), and being nominated for the "2007

service, polling Confirmit for the availability of new data (e.g., new training data, new codes in an existing codeframe, new verbatim data in need of coding, etc.), fetching these data from Confirmit, processing them, and uploading them back into Confirmit for reporting and later use.

In mid 2008 VCS™ was integrated into Language Logic's Ascribe™ platform, where it is now available to all Ascribe™ customers. We will refer to this version of VCS™ as version 2.0; this has numerous enhancements with respect to VCS™ 1.0, including support for verbatim data in languages other than English (the languages currently supported are English, Spanish, French, German, and Italian), accuracy and accuracy trend estimation, support for proactive validation, and others; all these features will be described more in detail in the next sections.

2.2. The machine learning approach to automated verbatim coding

VCS™ is an adaptive system for automatically coding verbatim responses under *any* user-specified codeframe; given such a codeframe, VCS™ *automatically* generates an automatic coder for this codeframe.

Actually, the basic unit along which VCS™ works is not the codeframe but the *code*: given a codeframe consisting of several codes, for each such code VCS™ automatically generates a *binary coder*, i.e., a system capable of deciding whether a given verbatim should or should not be attributed the code. The consequence of this mode of operation is that all binary coders are applied to the verbatim independently of each other, and that zero, one, or several codes can be attributed to the same verbatim (however, see Question 5 in Section 3 for exceptions).

VCS is based on a *learning metaphor*, according to which the system learns from manually coded data the characteristics that a new verbatim should have in order to be attributed the code; the manually coded verbatims that are fed to the system for the purpose of generating the binary coders are called the *training verbatims*. The training verbatims need to include *positive examples* of the code (i.e., verbatims to which a human coder has attributed the code) and *negative examples* of the code (i.e., verbatims to which a human coder has decided not to attribute the code): by examining both, the system identifies the *discriminating characteristics* of the verbatims, i.e., the characteristics that will help the binary coder in deciding whether to attribute the

Technology Effectiveness Award" from the Association for Survey Computing. The paper describing it (Macer et al., 2007) was also nominated for "Best Paper Award" at the 50th Annual Conference of the Market Research Society.

code or not to a yet uncoded verbatim. In other words, by examining the training verbatims the system generates a "mental model" of what it takes for a verbatim to be attributed the code; once these mental models (namely, the binary coders) are generated, they can be applied to coding yet uncoded verbatims (from now on the yet uncoded verbatims which are automatically coded by the binary coders will be called *test verbatims*). It is typically the case that, in a real application, the training verbatims will be much fewer than the test verbatims, so that the human coder, after coding a small portion of a survey and training the system with it, will have the binary coders code automatically the remaining large part of the survey.

In practice, it is not the case that training proceeds on a code-by-code basis: in VCS™ a user wanting to provide training examples to the system typically reads a verbatim and attributes to it the codes she deems fit; the intended meaning is that for all the codes in the codeframe that she does not attribute to the verbatim, the verbatim is to be considered a negative example.

It is important to recognize that VCS™ does *not* attempt to learn how to code in an "objectively correct" fashion, since coding is an inherently subjective task, in which different coders often disagree with each other on a certain coding decision, even after attempts at reconciling their views. What VCS™ only learns to do is *to replicate the subjective behaviour of the human coder who has coded the training examples.* If two ore more coders have been involved in manually coding the training examples, each coding a separate batch of verbatims, than VCS™ will mediate between them, and its coding behaviour will be influenced by the subjectivities of them all. This means that it is of key importance to provide training examples that are reliably coded, if possible by an expert coder (however, see Section 4.2.1 for some computer assistance in this phase).

Advantages of the learning-based approach are that, unlike with several other computerised solutions for coding open-ended data:

– No domain-dependent dictionaries are involved; VCS™ can be called a "plug-and-play" system, where the only input necessary for the system to work is manually coded data for training.

– There is no need to pay experts for writing coding rules in some arcane (Boolean, fuzzy, probabilistic, etc.) language; what the system does is basically generating those rules automatically, without human intervention, from the training examples.

– It is easy to update the system to handle a revised codeframe, a brand new codeframe, or a brand new survey. If a user, after training the system, needs to add a new code to the codeframe, she only needs to add training examples for the new code (this may simply mean checking which of the previously coded examples have the new code too) and have the system generate the binary coder for the new code; the binary coders for the other codes are unaffected. If a user, after training the system, needs to set it up to work on an entirely new question, or an entirely new survey, she does not need to "reprogram" the system: she only needs to provide the appropriate training examples for the new question, or survey (on this, see also Question 6 in Section 3).

Note that VCS$^{\text{TM}}$ does not even look (i) at the question which has elicited the answer, and (ii) at the textual descriptions of the codes in the codeframe (the codes can thus be numerical codes with no apparent meaning).

A visual description of the process upon which VCS$^{\text{TM}}$ is based is given in Figure 1. The area at top left represents the *training* phase: a human coder manually codes a (typically small) sample of uncoded verbatims and feeds them to a "trainer" who then generates the binary coders. The area at the bottom represents the automatic *coding* phase: the binary coders generated in the training phase are fed to a general-purpose coding engine that, once fed with a (typically large) set of yet uncoded verbatims, automatically codes them. Now these verbatims are ready for use in reporting (bottom left) or for taking individual decisions based on the codes automatically attributed to the individual responses, such as calling up a customer whose response has been given the code "Very unhappy; may defect to the competition".

The area at top right represents a phase we have not discussed yet, i.e., the *validation* phase. After automatic coding has been performed the user may wish to take a look at some of the automatically coded verbatims and correct potential mistakes she spots. If she does so, the manually corrected verbatims may be used as further training examples, so that the system can be re-trained with an augmented training set. It turns out, unsurprisingly, that the re-trained binary coders tend to be more accurate than the previously generated ones, especially when coding verbatims that are somehow "similar" to the ones that have been manually corrected. More than one re-training iteration can be performed, depending on available humanpower and depending on whether the desired level of accuracy has been reached or not.

Usually, in a given iteration of the validation process the user will inspect and correct only a small portion of the automatically coded verbatims. VCS$^{\text{TM}}$ 2.0, upon returning the automatically coded verbatims, sorts them in terms of the *confidence* with which the binary

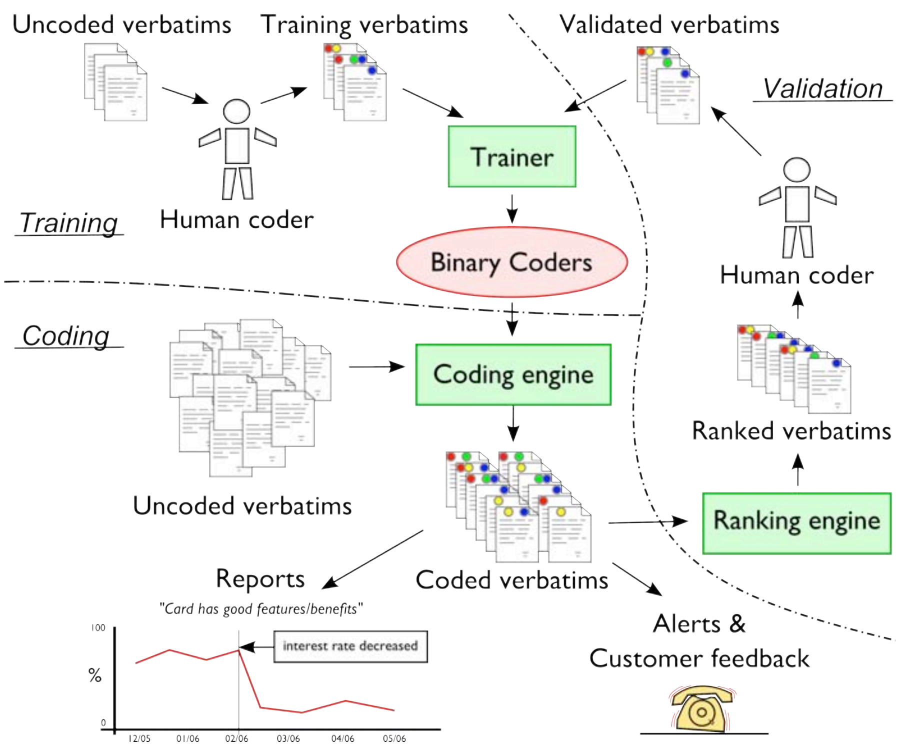

Figure 1. A visual description of the process upon which VCS^{TM} is based.

coders have coded them, so that the verbatims that VCS^{TM} has coded with the smallest confidence will be placed at the top of the list[14]. This has a double advantage, since the user is encouraged to first and foremost validate (i) the verbatims that have a higher chance of being miscoded, which allows the coder to remove mistakes from the result set; and (ii) the verbatims that, being problematic to VCS^{TM}, convey the largest amount of information to VCS^{TM} when provided as training examples.

3. Frequently asked questions

QUESTION 1. *Does VCS^{TM} attribute identical codes to identical verbatims?*

Yes, as it may be expected, two ore more identical copies of the same verbatim that are encountered at coding time are attributed exactly the

[14] This feature was not available in VCS^{TM} 1.0.

same codes. Unless, of course, the binary coders have been retrained after validation, or have been retrained with a different (e.g., augmented) set of training verbatims, in which case a verbatim that had been coded one way before retraining could be coded another way after retraining.

QUESTION 2. *What about multiple (consistently coded) training copies of the same verbatim? Will VCS^{TM} treat them as if a single copy had been included in the training set?*

No, VCS^{TM} is sensitive to the presence of multiple consistently coded training copies of the same verbatim; these have the effect of "reinforcing the point", e.g., telling the system that this verbatim is an *important* positive example of the codes that are attached to it. This case indeed occurs often, e.g., in "semi-open questions" from market research questionnaires, in which answers, albeit textual in nature, are typically very short, often consisting of one or two words. It certainly seem natural that verbatims frequently encountered in training should be treated in a "reinforced" way.

QUESTION 3. *If two inconsistently coded copies of the same verbatim are provided for training, does the system fall apart?*

Luckily not. VCS^{TM} was designed to handle these types of inconsistency, since it is frequently the case that inconsistently coded copies of the same verbatim are encountered at training time (especially if the training set has been generated by two or more human coders, each working independently of the others and each coding a separate batch of examples). VCS^{TM} implements a "fail soft" policy, such that, if two copies of verbatim v are encountered at training time, one a positive and the other a negative example of code c, the binary coder for code c will simply behave as it had been provided with neither of these two training examples: i.e., contradictory pieces of training information neutralize each other. The binary coders for codes other than c are obviously unaffected.

QUESTION 4. *If a given verbatim is provided as a training example, and an identical copy of it is encountered again, uncoded, in the coding phase, does this copy receive the same codes that the training copy has?*

Not necessarily. The "training" copy of the verbatim can have a set of codes, and the "uncoded" copy may be automatically attributed a different set of codes, since the coding behaviour of the system is determined, holistically, by all the training it has received, and not by a single training example. Were it not like this, of course, the system could not "fail soft" as discussed in the answer to Question 3.

QUESTION 5. *Does VCS^{TM} always attribute at least one code to each verbatim?*

No. In the default setting VCS$^{\text{TM}}$ may attribute several codes, one code, or no code at all to a given verbatim, since each code in the codeframe is treated in isolation of the others. If no code at all is attributed to the verbatim, this is akin to giving the verbatim the special code "Others" (which is indeed what VCS$^{\text{TM}}$ 1.0 did). Of course this is just cosmetics, since the interface of VCS$^{\text{TM}}$ 2.0 allows to easily spot the verbatims with no code attached anyway.

However, the user may change this default setting on a question-by-question basis, i.e., for a given question the user may specify that one and only one of the following constraints should be enforced: i) at least n codes must be attached to a verbatim; ii) at most n codes must be attached to a verbatim; iii) exactly n codes must be attached to a verbatim. The value of n is 1 by default but can be changed at will.

QUESTION 6. *I have trained the system to work for a certain question, using a certain codeframe. Now I need to set it up to work on a different question, but using the same codeframe as before. Can I use for this new question the binary coders I had trained on the same codeframe for the previous question?*

Yes, or sort of. The best course of action is to use the previously generated binary coders and then to engage in an accurate validation effort. The reason is that there are both commonalities and differences in meaning between the same code as used in two different contexts. It is our experience that the commonalities are stronger than the differences, so this suggests that we should leverage on the previously generated binary coders; however, careful validation will smooth out the differences, and will attune the binary coders to the meaning that the codes take up in the new context. In sum, if the manual effort you can afford is, say, manually coding 500 (or even less) verbatims, you can certainly start from scratch, but it is probably better if you run the binary coders previously generated for the same codeframe on your verbatims and then validate 500 of the automatically classified ones.

QUESTION 7. *Does VCS^{TM} cater for sentiment-related codes / distinctions, such as "Positive" and "Negative"?*

It certainly does. VCS$^{\text{TM}}$ was designed with survey research in mind, i.e., with the awareness that surveys not always attempt to capture purely topic-related distinctions, but often attempt to capture notions that have to do with the emotions, sentiments, and reactions of respondents. Indeed, one of the main backgrounds of the designers of VCS™ is sentiment analysis and opinion mining. See Section 5 for more details.

QUESTION 8. *How much time and money am I going to save on a project by using VCS™?*

The answer depends very much on how many training examples are provided, how many verbatims are then coded automatically, how high is the per-verbatim cost of human coding, whether fast turnaround of results is important to you, how accurate the resulting system is required to be, and many other variables. A detailed study on practical benefits found in operation was run by the Egg banking group on VCS™ 1.0; the results of this study can be found in (Macer et al., 2007, p. 15).

4. Other and upcoming features

Before concluding, we briefly hint at a couple of other features that VCS™ 2.0 is endowed with, and at some further features that we have recently developed and that will be integrated in the next release of VCS™.

4.1. Other features in VCS™ 2.0

4.1.1. *Robustness to ill-formed input*

One VCS™ 2.0 feature which is of particular interest in customer satisfaction and market research applications is robustness to orthographically, syntactically, or other type of, ill-formed input. Certainly, we cannot expect verbatims to be conveyed in spotless English (or other language), since in most cases verbatim text is produced carelessly, and by casual users who are anything but professional writers. So, a system which requires verbatim text to conform to the standard rules of English grammar and to be free of typographical errors would be doomed to failure in this application context. VCS™ is instead devised to be robust to the presence of ill-formed input; to witness, the results of Table I in (Esuli et al., 2009) were obtained on datasets of authentic verbatim text, and are indeed fraught with ill-formed text of any type. Indeed, VCS™ *learns* to deal with ill-formed verbatims from ill-formed training verbatims; in other words, once the binary coders are trained from a training set that itself contains ill-formed input, they will tend to outperform, in coding yet uncoded ill-formed verbatims, binary coders that have instead been trained on well-formed input!, since, upon coding, they will encounter ill-formed linguistic patterns they have encountered in the training stage (e.g., common abbreviations, common syntactic mistakes, common typos, slang, idioms, etc.).

4.1.2. *Dynamic estimation of current and future accuracy levels*

A second VCS$^{\mathrm{TM}}$ 2.0 feature we have not yet discussed has to do with letting the user know what accuracy she can expect from VCS$^{\mathrm{TM}}$ on a given set of verbatims, given the amount of training she has performed already, and letting her know whether it is likely that there are still margins of improvement by doing further training or validation. Indeed, once the user submits a set of training examples, VCS$^{\mathrm{TM}}$ generates the binary coders, and returns the (F_1, PD_M, PD_A) values computed by 10-fold cross-validation on the training set: these figures thus represent an estimate (actually: a pessimistic estimate, given that only 90% of the training set has been used for training in each of the 10 experiments) of the accuracy that the generated binary coders will display on yet uncoded verbatims. Moreover, VCS$^{\mathrm{TM}}$ computes (F_1, PD_M, PD_A) by 10-fold cross-validation on 80% of the training set, and returns to the user the percentile difference between this triplet of results and the previously mentioned triplet; this difference thus represents the difference in performance that providing the other 20% of training brought about, and can thus be interpreted as the current "accuracy trend", i.e., an indication of whether performing further training or validation is still going to bring about an improvement in accuracy, and to what degree.

4.1.3. *Mock verbatims*

Even when the training set is reasonably sizeable it often happens that, while for some codes there are lots of training examples, other codes are heavily undersubscribed, to the point that for some of them there may be a handful, or even no training examples at all. For these latter codes, generating an accurate binary coder is hardly possible, if at all. In these cases, hand-coding other verbatims with the intent of providing more training data for the underpopulated codes may be of little help, since examples of these codes tend to occur rarely.

In this case, VCS$^{\mathrm{TM}}$ 2.0 allows the user to define "mock" training verbatims for these codes, i.e., examples of these codes that were not genuinely provided by a respondent but are made up by the user. In other words, VCS$^{\mathrm{TM}}$ allows the user to provide examples of "what a verbatim that would be assigned this code might look like". Such a mock verbatim may take two forms, i.e., (1) a verbatim that is completely made up by the user, or (2) a verbatim that the user has cropped away from some other genuine, longer verbatim. VCS$^{\mathrm{TM}}$ treats mock verbatims differently from authentic ones, since (i) it flags them as nonauthentic, so that they do not get confused with the authentic ones for reporting and other similar purposes, and (ii) it gives them extra weight in the training phase, since it deems that their very nature makes them "paradigmatic" examples of the codes that are attached to them.

In order to minimize the user's work in defining mock verbatims, given a training set VCS^{TM} presents to the user a list of the codes in the codeframe ranked in increasing number of associated training examples, so that the user may indeed concentrate in defining mock verbatims for the codes that are more in need of them

4.2. Upcoming features

4.2.1. *Training data cleaning*

Among the forthcoming features, the first is a facility for *training data cleaning.* In many situations it is actually the case that there might be miscodings in the verbatims that are being provided to the system for training purposes; this might be a result of the fact that junior, inexperienced coders have done the coding, of the fact that the coding had been done under time pressure, or other. Of course, "bad data in, bad data out", i.e., you cannot expect the system to code accurately if it has been trained with inaccurate data. As a result, a user might wish to have a computerised tool that helps her in "cleaning" the training data, i.e., in identifying the miscoded training verbatims so as to be able to correct them. Of course, such a tool should make it so that she does not need to revise the *entire* set of training examples.

The tool we have implemented and that we are now testing returns to the user, as a side-effect of training, a sorted list of the training verbatims, sorted in order of decreasing likelihood that the verbatim was indeed miscoded. This allows the user to revise the training set starting from the top of the list, where the "bad" examples are more likely to be located, working down the list until she sees fit. The tool works on a code-by-code basis, i.e., for any given code the tool returns a list of examples sorted in decreasing likelihood that the verbatim was indeed miscoded *for this code* (i.e., it is either a false positive or a false negative for this code). This allows the user to perform *selective* cleaning, e.g., perform a cleaning operation for those codes whose binary coders have not yet reached the desired level of accuracy, and forget about codes on which VCS^{TM} is already performing well.

How does the tool work? The basic philosophy underlying it is that a training verbatim has a high chance of having been miscoded when the codes manually attributed to it are highly *at odds* with the codes manually attributed to the other training verbatims. For instance, a training verbatim to which a given code has been attributed and which is highly similar to many other verbatims to which the same code has not been attributed, is suspect, and will thus be placed high in the sorted list for that code. In other words, VCS^{TM} has a notion of the *internal consistency* of the training set, and the tool sorts the training verbatims in terms of how much they contribute to disrupting this internal consistency.

4.2.2. *Computer-assisted codeframe generation*

The second feature currently under development is *computer-assisted codeframe generation*. Codeframe generation is a time-consuming operation in survey management, and one that requires expertise. While it is not feasible to think that an expert survey specialist would entirely defer the codeframe generation activity to a computerised tool, it is certainly the case that she might welcome a tool that *helps* her in generating the codeframe, by suggesting an initial codeframe that she can then revise and tune.

The tool that we have implemented and that we are now testing receives from the user a set of verbatims and an indicative number n of codes the user would like the codeframe to contain; as a result, the tool partitions the verbatims into n groups (i.e., each verbatim is placed in one and only one group) in which the pairwise similarity between verbatims belonging to the same group is maximized while the pairwise similarity between verbatims belonging to different groups is minimized. For each group the tool also returns a description consisting of a set of representative words or phrases; e.g., in a customer satisfaction survey the set {“fed up”, “bad service”, “awful”, “competing product”, “switch”} might suggest to the user that this group is about respondents threatening to switch to a competing product.

At this point the user might realize that she would like more granularity, and ask the number n of groups to be increased: in this case the tool splits some of the already generated groups into subgroups and generates a description for the newly generated subgroups. Conversely, the user might realize that she would like less granularity, and ask the number n of groups to be decreased: in this case the tool merges some of the already generated groups into supergroups and generates a description for the newly generated supergroups. When the desired granularity is reached, if the user is not satisfied with the end result she may want to revise the codeframe, adding or removing or renaming groups, or moving verbatims from one group to the other, until the final codeframe is generated.

4.2.3. *Support for hierarchical codeframes*
A third feature currently under development is the support for hierarchically organized codeframes. Currently, VCS™ only deals with "flat" codeframes, i.e., codeframes consisting of a plain set of codes. The tool that we have implemented and that we are now testing supports codeframes structured as trees of supercodes (e.g., "Soft drink") and subcodes (e.g., "Coke"). At training time, all training examples of the subcodes will also be, by definition, training examples of the corresponding supercode. At coding time, a given verbatim will be fed to the binary coder for the subcode only if the binary coder for the corresponding supercode has returned a positive decision; this will ultimately bring about an exponential increase in coding efficiency, since for each verbatim to code entire subtrees will be discarded from consideration, thus allowing VCS™ to operate speedily even on codeframes consisting of several thousands codes.

5. Concluding remarks

Before concluding, note that we have not given many details on the internal workings of VCS™. This is partly due to the commercial nature of the VCS™ project, and partly due to the fact that a description of these workings, to any level of detail, would likely distract the reader from understanding the important facts about VCS™, i.e., how a researcher should use it and what she should expect from it. The interested reader may anyhow reconstruct, if not all details, the main tracts of how VCS™ works by looking at the authors' published research on issues of text analysis for meaning and opinion extraction (Baccianella et al., 2009; Debole and Sebastiani, 2003; Esuli and Sebastiani, 2009b; Nardiello et al., 2003), learning algorithms for text classification (Sebastiani et al., 2000; Esuli et al., 2008), and opinion mining for sentiment analysis (Argamon et al., 2007; Esuli and Sebastiani, 2005; Esuli and Sebastiani, 2006a; Esuli and Sebastiani, 2006b; Esuli and Sebastiani, 2007a; Esuli and Sebastiani, 2007b). In particular, the tool used in the validation phase (Section 2.2) draws inspiration from basic research reported in (Esuli and Sebastiani, 2009a); the training data cleaning tool (Section 4.2.1) is based on the very recent (Esuli and Sebastiani, 2009c); the tool for computer-assisted codeframe generation (Section 4.2.2) is based on previous research described in (Geraci et al., 2008); and the support for hierarchical codeframes is based on insights obtained in (Fagni and Sebastiani, 2007; Esuli et al., 2008).

Overall, we think that the VCS™ system we have presented has the potential to revolutionize the activity of coding open-ended responses as we know it today, since (i) it allows a user to autonomously create automatic coding systems for *any* user-specified codeframe and for *any* type of survey conducted (as of now) in any of five major European languages, with no need for specialized dictionaries or domain-dependent resources; (ii) it allows to improve the accuracy of the generated coding systems almost at will, by validating, through a convenient interface, carefully selected samples of the automatically coded verbatims. Even more importantly, for doing any of the above it requires on the part of the user no more skills than ordinary coding skills.

In a forthcoming, companion paper (Esuli et al., 2009) we will show, by discussing a number of experiments we have run on several datasets of real respondent data, how the binary coders generated by VCS™ are characterized by very good accuracy and excellent training and coding speed.

Acknowledgements

Many people have contributed to the VCS™ project in the course of these last 7 years. Daniela Giorgetti is to be credited with first bringing the very existence of "survey coding" to the attention of the third author, and for first creating with him VCS™ version 0.1, for designing and implementing which Irina Prodanof provided partial funding and for testing which Tom Smith provided datasets. Ivano Luberti has been instrumental in turning all of the ideas discussed in this paper into an industrial-strength software package. Mark Pearson, in his then-capacity of Manager of Customer Insight at Egg plc, foresaw the potential of the machine learning approach for verbatim coding and had Egg plc invest on it. Tim Macer, on behalf of Egg plc, has been a constant source of advice, feedback and ideas throughout the duration of the project that led VCS™ 1.0 to be deployed on Egg's IT infrastructure. Charles Baylis has provided feedback and inspiration throughout the project that has led to VCS™ 2.0 and to its integration in the Ascribe™ package. Thanks are also due to Lee Birkett, Rudy Bublitz, John Jennick, Dean Kotcha, Fabio Nanni, Tara O'Hara, Antonio Paduano, Rino Razzi, Lara Rice, Carol Sheppard, Chrissy Stevens, and Rich Thoman, for useful discussions, and for ultimately contributing to making all of this happen.

References

Argamon, S., K. Bloom, A. Esuli, and F. Sebastiani: 2007, 'Automatically Determining Attitude Type and Force for Sentiment Analysis'. In: *Proceedings of the 3rd Language Technology Conference (LTC'07).* Poznań, PL, pp. 369–373.

Baccianella, S., A. Esuli, and F. Sebastiani: 2009, 'Multi-Facet Rating of Product Reviews'. In: *Proceedings of the 31st European Conference on Information Retrieval (ECIR'09).* Toulouse, FR, pp. 461–472.

Debole, F. and F. Sebastiani: 2003, 'Supervised term weighting for automated text categorization'. In: *Proceedings of the 18th ACM Symposium on Applied Computing (SAC'03).* Melbourne, US, pp. 784–788.

Duda, R. O., P. E. Hart, and D. G. Stork: 2001, *Pattern Classification.* New York, US: John Wiley & Sons, 2nd edition.

Esuli, A., T. Fagni, and F. Sebastiani: 2008, 'Boosting Multi-Label Hierarchical Text Categorization'. *Information Retrieval* **11**(4), 287–313.

Esuli, A., T. Fagni, and F. Sebastiani: 2009, 'Machines that Learn how to Code Open-Ended Survey Data. Part II: Experiments on Real Respondent Data'. *International Journal of Market Research.* Submitted for publication.

Esuli, A. and F. Sebastiani: 2005, 'Determining the semantic orientation of terms through gloss analysis'. In: *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM'05).* Bremen, DE, pp. 617–624.

Esuli, A. and F. Sebastiani: 2006a, 'Determining Term Subjectivity and Term Orientation for Opinion Mining'. In: *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL'06).* Trento, IT, pp. 193–200.

Esuli, A. and F. Sebastiani: 2006b, 'SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining'. In: *Proceedings of the 5th Conference on Language Resources and Evaluation (LREC'06).* Genova, IT, pp. 417–422.

Esuli, A. and F. Sebastiani: 2007a, 'PageRanking WordNet synsets: An application to Opinion Mining'. In: *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL'07).* Prague, CZ, pp. 424–431.

Esuli, A. and F. Sebastiani: 2007b, 'Random-Walk Models of Term Semantics: An Application to Opinion-Related Properties'. In: *Proceedings of the 3rd Language Technology Conference (LTC'07).* Poznań, PL, pp. 221–225.

Esuli, A. and F. Sebastiani: 2009a, 'Active Learning Strategies for Multi-Label Text Classification'. In: *Proceedings of the 31st European Conference on Information Retrieval (ECIR'09).* Toulouse, FR, pp. 102–113.

Esuli, A. and F. Sebastiani: 2009b, 'Encoding Ordinal Features into Binary Features for Text Classification'. In: *Proceedings of the 31st European Conference on Information Retrieval (ECIR'09).* Toulouse, FR, pp. 771–775.

Esuli, A. and F. Sebastiani: 2009c, 'Training Data Cleaning for Text Classification'. Technical Report 2009-TR-005, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche.

Fagni, T. and F. Sebastiani: 2007, 'On the Selection of Negative Examples for Hierarchical Text Categorization'. In: *Proceedings of the 3rd Language & Technology Conference (LTC'07).* Poznań, PL, pp. 24–28.

Geraci, F., M. Maggini, M. Pellegrini, and F. Sebastiani: 2008, 'Cluster Generation and Labeling for Web Snippets: A Fast, Accurate Hierarchical Solution'. *Journal of Internet Mathematics* **3**(4), 413–444.

Giorgetti, D. and F. Sebastiani: 2003, 'Automating Survey Coding by Multiclass Text Categorization Techniques'. *Journal of the American Society for Information Science and Technology* **54**(14), 1269–1277.

Hastie, T., R. Tibshirani, and J. H. Friedman: 2001, *The Elements of Statistical Learning*. Heidelberg, DE: Springer Verlag.

Macer, T.: 1999, 'Designing the survey of the future'. *Research* (395).

Macer, T.: 2000, 'Making coding easier'. *Research* (407).

Macer, T.: 2002, 'Ascribe from Language Logic'. *Quirk's Marketing Research Review* **16**(7).

Macer, T.: 2007a, 'Coding-Modul Reviewed'. *Research* (490).

Macer, T.: 2007b, 'Voxco Command Center'. *Quirk's Marketing Research Review* **21**(1).

Macer, T.: 2008, 'WordStat from Provalis Research'. *Research* (507).

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.

Manning, C. D., P. Raghavan, and H. Schütze: 2008, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press.

Mitkov, R. (ed.): 2003, *The Oxford handbook of computational linguistics*. Oxford University Press.

Nardiello, P., F. Sebastiani, and A. Sperduti: 2003, 'Discretizing Continuous Attributes in AdaBoost for Text Categorization'. In: F. Sebastiani (ed.): *Proceedings of the 25th European Conference on Information Retrieval (ECIR'03)*. Pisa, IT, pp. 320–334.

O'Hara, T. J. and T. Macer: 2005, 'Confirmit 9.0 reviewed'. *Research* (465).

Pang, B. and L. Lee: 2008, 'Opinion Mining and Sentiment Analysis'. *Foundations and Trends in Information Retrieval* **2**(1/2), 1–135.

Reja, U., K. L. Manfreda, V. Hlebec, , and V. Vehovar: 2003, 'Open-ended vs. Close-ended Questions in Web Questionnaires'. In: A. Ferligoj and A. Mrvar (eds.): *Developments in Applied Statistics*. Ljubljana, SL: Faculty of Social Sciences, University of Ljubljana, pp. 159–177.

Schuman, H. and S. Presser: 1979, 'The Open and Closed Question'. *American Sociological Review* **44**(5), 692–712.

Sebastiani, F., A. Sperduti, and N. Valdambrini: 2000, 'An improved boosting algorithm and its application to automated text categorization'. In: *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM'00)*. McLean, US, pp. 78–85.

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