Optimising Human Inspection Work in Automated Verbatim Coding

Giacomo Berardi, Andrea Esuli, and Fabrizio Sebastiani[†]
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
56124 Pisa, Italy
Email: firstname.lastname@isti.cnr.it

Abstract. Automatic verbatim coding technology is essential in many contexts in which, either because of the sheer size of the dataset we need to code, or because of demanding time constraints, or because of cost-effectiveness issues, manual coding is not a viable option. However, in some of these contexts the accuracy standards imposed by the customer may be too high for today's automated verbatim coding technology; this means that human coders may need to devote some time to inspecting (and correcting where appropriate) the most problematic autocoded verbatims, with the goal of increasing the accuracy of the coded set. We discuss a software tool for optimising the human coders' work, i.e., a tool that minimizes the amount of human inspection required to reduce the overall error down to a desired level, or that (equivalently) maximises the reduction in the overall error achieved for an available amount of human inspection work.

1. Introduction

In the last 10 years we have championed an approach to automatically coding open-ended answers ("verbatims") based on "machine learning" (Giorgetti and Sebastiani, 2003). Based on these principles we have built an automated verbatim coding system which we have variously applied to coding surveys in the social sciences (Giorgetti et al., 2003), in customer relationship management (Macer et al., 2007), and in market research (Esuli and Sebastiani, 2010).

This system (see Figure 1) is based on a *supervised learning metaphor*: the system learns, from sample manually coded verbatims (*training examples*), the characteristics a new uncoded verbatim (a *test example*) should have in order to be attributed a given code; the human operator who feeds the training examples to the system plays the role of the "supervisor" (Alpaydin, 2010; Mohri et al., 2012).

The machine learning approach to automated verbatim coding has shown very good accuracy in many real-world studies (see (Esuli and Sebastiani, 2010) for examples). However, there may indeed be scenarios in which the accuracy standards imposed by the customer (e.g., as specified in a Service Level Agreement) are too demanding, not only for this approach but for any existing automated verbatim coding technology. If full manual coding by expert coders is not a viable option (due to the sheer size of the dataset that needs coding, or to demanding time constraints, or to cost issues, or to a combination of all these),

[†] The order in which the authors are listed is purely alphabetical; each author has given an equally important contribution to this work.

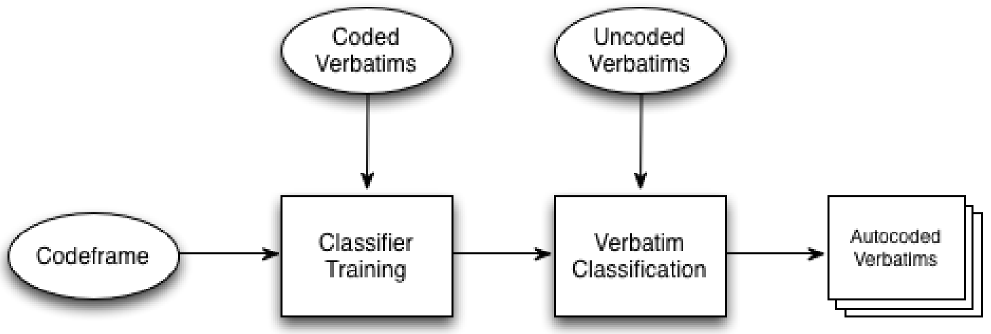

Figure 1. Architecture of a verbatim coding system based on supervised machine learning.

a possible strategy may consist in coding the data by means of an automatic method, and then having one or more human coders inspect (and correct where appropriate) the most problematic among the automatically coded verbatims[1].

In this paper we will be interested in application scenarios of the latter kind. Specifically, the task we will set ourselves will be that of devising software tools that support the post-coding inspection work by the human coders. For us, supporting the coders' work will mean *maximising the cost-effectiveness* of their work; in other words, we will be interested in software methods that minimise the amount of human inspection work required to reduce the overall error in the data down to a certain level, or that (equivalently) maximise the reduction in the overall error achieved for a certain amount of human coder's inspection work.

2. A Worked-out Example

In order to see how human coders may be effectively supported in their post-editing work, let us look at a specific example. Let us assume that the coding task consists in deciding whether a given code applies or not to any of a set of uncoded verbatims; coding according to an entire codeframe is qualitatively analogous, since the process above can be repeated for each code in the codeframe.

Let us also assume that a set of uncoded verbatims have been automatically coded; for simplicity of illustration we here assume that this set consists of 20 verbatims only. We can measure the accuracy obtained in this automatic coding job by (a) choosing an accuracy measure, (b) filling out a contingency table, and (c) evaluating the chosen measure on this table. For illustration purposes we assume that our accuracy measure is the well-known F_1 measure (see (Esuli and Sebastiani, 2010) for a detailed discussion), defined as

$$F_1 = \frac{2 \cdot TP}{(2 \cdot TP) + FP + FN} \tag{1}$$

[1] In the rest of this paper we will simply write "inspect" to actually mean "inspect and correct where appropriate".

where by TP, FP, FN, TN, we indicate, as customary, the number of true positives, false positives, false negatives, true negatives, derived from the automatic coding; F_1 values range from 0 (worst) to 1 (best).

Figure 2 depicts a situation in which the automated coding process has returned 4 true positives, 3 false positives, 4 false negatives, and 9 true negatives, resulting in a value of $F_1 = \frac{2 \cdot 4}{(2 \cdot 4)+3+4} = 0.533$. The 20 verbatims are represented in the two rows at the bottom via green and red cards; the upper row represents the coding decisions of the system ("predictions"), while the lower row represents the correct decisions that an ideal system would have taken. A green card represents a "yes" (the verbatim has the code), while a red card represents a "no" (the verbatim does not have the code); a correct decision is thus represented by the upper and lower card in the same column having the same colour[2].

		predicted	
		Y	**N**
true	**Y**	$TP = 4$	$FP = 3$
	N	$FN = 4$	$TN = 9$

$$F_1 = \frac{2TP}{2TP + FP + FN} = 0.533$$

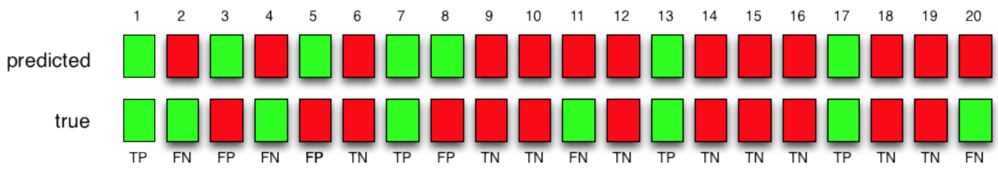

Figure 2. A worked-out example, representing a contingency table (upper left part of the figure) deriving from the automatically coded examples (lower part) and from which accuracy is computed (formula in the upper right part).

Let us imagine a scenario in which the customer insists that the data must be coded with an accuracy level of at least $F_1 = 0.800$. In this case, after checking that the value of F_1 that the automatic classifier has obtained is 0.533, the human coder decides to inspect some of the verbatims until the desired level of accuracy has been obtained[3]. Let us assume that the coder examines the verbatims at the bottom of Figure 2 in left-to-right order. The first verbatim that the coder examines is a true positive; no correction needs to be done, the value of F_1 is unmodified, and the coder moves on to the second verbatim. This

[2] Note that this example is artificial, and is for illustration purposes only. In the real-world coding studies reported in (Esuli and Sebastiani, 2010) our automated verbatim coding system has always obtained higher F_1 values, ranging from 0.55 to 0.92 depending on the number of training examples available and on the inherent difficulty of the coding task.

[3] Actually, the human coder does *not* know the level of accuracy that the automatic classifier has obtained, since she does not know the true code assignments of the verbatims. However, this level of accuracy can be at least *estimated* in real time, via a technique called "10-fold cross-validation" that will be discussed in Sections 4.1.4 and 7.

is a false negative, and correcting it decreases FN by 1 and increases TP by 1, which means that F_1 now becomes $F_1 = \frac{2 \cdot 5}{(2 \cdot 5)+3+3} = 0.625$. The third is a false positive, and correcting it decreases FP by 1 and increases TN by 1, which means that F_1 now becomes $F_1 = \frac{2 \cdot 5}{(2 \cdot 5)+2+3} = 0.667$. The fourth is a false negative, which brings about $F_1 = \frac{2 \cdot 6}{(2 \cdot 6)+2+2} = 0.750$, and the fifth is a false positive, which yields $F_1 = \frac{2 \cdot 6}{(2 \cdot 6)+1+2} = 0.800$. At this point, having reached the minimum level of accuracy required by the customer, the human coder's task is over.

3. Ranking the Automatically Coded Verbatims

In the example illustrated in the previous section, bringing F_1 from 0.533 up to 0.800 has required the inspection of 5 verbatims, i.e., 25% of the entire set. Could the human coder have achieved the same improvement in accuracy with a smaller effort (i.e., by inspecting fewer verbatims)? Could she, by putting in the same effort, have reached a level of accuracy higher than $F_1 = 0.800$? The answer to both questions is yes, and the key to doing better is the order in which the verbatims are inspected.

For instance, the fact that the first inspected verbatim was a true positive was suboptimal. Inspecting verbatims that have been coded correctly is, for the human coder, wasted time, since no correction is performed and F_1 remains thus unmodified. Of course, there is no way to know in advance if the verbatim has been coded incorrectly or not. However, it would at least be desirable to know how *likely* it is that the verbatim has been coded incorrectly, i.e., to know its *probability of misclassification*; in this case, the system might rank the automatically coded verbatims in such a way as to top-rank the verbatims that have the highest such probability.

A second fact that jumps to the eye in the worked-out example of the previous section is that the increase in accuracy (i.e., the *gain*) determined by the correction of a false negative is higher (sometimes much higher) than the gain determined by the correction of a false positive. For instance, in correcting the second verbatim (a false negative) F_1 jumped from 0.533 to 0.625 (a +17.1% relative increase), while in correcting the third verbatim (a false positive) F_1 only moved from 0.625 to 0.667 (a mere +6.6% relative increase). This is not an idiosyncrasy of the F_1 measure, since for many accuracy measures the gain deriving from the correction of a false positive is different than the one deriving from the correction of a false negative[4]. So, the fact that two false positives were inspected and corrected while two false negatives were left uninspected and uncorrected (in 11th and 20th position, respectively) was also suboptimal.

[4] Note that this asymmetry holds despite the fact that F_1 pays equal attention to the ability of the system to avoid false positives (known as *precision*, and defined as $\pi = \frac{TP}{TP+FP}$) and to the ability of the system to avoid false negatives (known as *recall*, and defined as $\rho = \frac{TP}{TP+FN}$); in fact, $F_1 = (2 \cdot \pi \cdot \rho)/(\pi + \rho) = (2 \cdot TP)/((2 \cdot TP) + FP + FN)$. In other measures that pay, e.g., more attention to recall than to precision, the difference between the gain obtained by correcting a false positive and the gain obtained by correcting a false negative is amplified.

This means that the system should, other things being equal, rank higher those verbatims (the false negatives, in our case) that bring about a higher gain when corrected. In the example of the previous section, had we ranked the four false negatives at the top four rank positions and a false positive at the fifth, the same amount of human coder inspection work would have brought about an increase in F_1 from 0.533 to 0.889 (instead of 0.800); had we instead been happy with reaching $F_1 = 0.800$, the human coder would have reached it by inspecting and correcting only the *four* top-ranked verbatims (actually, by doing this she would have reached $F_1 = 0.842$).

In sum, we have learnt two key facts.

The first fact is that the order in which the human coder inspects the automatically coded verbatims is what determines the cost-effectiveness of her work. This fact should come as no surprise: the task of ranking a set of digital objects in terms of perceived usefulness to a given task is of paramount importance in nowadays' computer science as a whole, as perfectly exemplified by todays' search engines[5]. The task of ranking the automatically coded verbatims with the goal of maximising the cost-effectiveness of a human coder who scans (inspecting and correcting) the ranked list down to a certain depth, has been called *Semi-Automatic Text Classification* (SATC – (Berardi et al., 2012; Martinez-Alvarez et al., 2012)), to reflect the fact that it attempts to optimise a pipeline in which human and machine cooperate in achieving the goal of accurate classification.

The second fact we have learnt is that, if we want to order these verbatims so as to maximise the cost-effectiveness of the human coder's work, we should take two main factors into account, i.e., (a) the *probability* of misclassification of a given verbatim, and (b) the *gain* in coding accuracy that the verbatim brings about once inspected and corrected.

4. Utility Theory

What kind of mathematical theory should we use in order to devise such a ranking function?

The need to account for probabilities and gains immediately evokes *utility theory*, an extension of probability theory that incorporates the notion of *gain* (or *loss*) that accrues from a given course of action (Anand, 1993; von Neumann and Morgenstern, 1944). Utility theory is a general theory of rational action under uncertainty, and as such is used in many fields of human activity. For instance, utility theory is of paramount importance in betting, since in placing a certain bet we take into account (a) the probabilities of occurrence that we subjectively attribute to a set of outcomes (say, to the possible outcomes of the Arsenal FC vs. Chelsea FC game), and (b) the gains or losses that we obtain, having bet on one of them, if the various outcomes materialise.

[5] The main factor that, in the late '90s, decreed the success of Google over its then-competitors (e.g., AltaVista, Inktomi, etc.) was exactly its superior ranking function. Nowadays it is fair to say that the ranking function that Google uses is as secret as the recipe of Coca Cola.

In order to explain our method let us introduce some basics of utility theory. Given a set $A = \{\alpha_1, ..., \alpha_m\}$ of possible courses of action and a set $\Omega = \{\omega_1, ..., \omega_n\}$ of mutually disjoint events, the *expected utility* $U(\alpha_j, \Omega)$ that derives from choosing course of action α_j given that any of the events in Ω may occur, is defined as

$$U(\alpha_j, \Omega) = \sum_{\omega_i \in \Omega} P(\omega_i) G(\alpha_j, \omega_i) \tag{2}$$

where $P(\omega_i)$ is the *probability* of occurrence of event ω_i, and $G(\alpha_j, \omega_i)$ is the *gain* obtained if, given that α_j has been chosen, event ω_i occurs. For instance, α_j may be the course of action "betting on Arsenal FC's win" and Ω may be the set of mutually disjoint events $\Omega = \{\omega_1, \omega_2, \omega_3\}$, where ω_1="Arsenal FC wins", ω_2="Arsenal FC and Chelsea FC tie", and ω_3="Chelsea FC wins"; in this case,

- $P(\omega_1)$, $P(\omega_2)$, and $P(\omega_3)$ are the probabilities of occurrence that we subjectively attribute to the three events ω_1, ω_2, and ω_3;
- $G(\alpha_j, \omega_1)$, $G(\alpha_j, \omega_2)$, and $G(\alpha_j, \omega_3)$ are the economic rewards we obtain, given that we have chosen course of action α_j (i.e., given that we have bet on the win of Arsenal FC), if the respective event occurs. Of course, our economic reward will be positive if ω_1 occurs and negative if either ω_2 or ω_3 occur.

When we face alternative courses of action, acting rationally means choosing the course of action that maximises our expected utility. For instance, given the alternative courses of action α_1="betting on Arsenal FC's win", α_2="betting on Arsenal FC's and Chelsea FC's tie", α_3="betting on Chelsea FC's win", we should pick among $\{\alpha_1, \alpha_2, \alpha_3\}$ the course of action that maximises $U(\alpha_j, \Omega)$.

4.1. Ranking Automatically Coded Verbatims via Utility Theory

How does this translate into a method for ranking automatically coded verbatims? Assume we have a set $D = \{d_1, ..., d_n\}$ of automatically coded verbatims that we want to rank, and assume that F_1 is our evaluation measure. For instantiating Equation 2 concretely we need

1. to decide what our set $A = \{\alpha_1, ..., \alpha_m\}$ of alternative courses of action is;
2. to decide what the set $\Omega = \{\omega_1, ..., \omega_n\}$ of mutually disjoint events is;
3. to specify how we compute their probabilities of occurrence $P(\omega_i)$;
4. to define the gains $G(\alpha_j, \omega_i)$.

Let us discuss each of these steps in turn[6].

[6] The method we are going to discuss in this section is an improved variant of a method that has already been presented in (Berardi et al., 2012) in much greater mathematical detail; an extended version is in preparation (Berardi et al., 2014). The goal of the present paper is

4.1.1. *Courses of Action*

Concerning Step 1, we will take the action of inspecting (and correcting, if needed) verbatim d_j as course of action α_j. In this way we will evaluate the expected utility $U(d_j, \Omega)$ (i.e., the expected increase in overall accuracy) that derives from inspecting each verbatim d_j, and we will be able to rank the verbatims by their $U(d_j, \Omega)$ value, so as to top-rank the ones with the highest expected utility.

4.1.2. *Events*

Concerning Step 2, we have argued that the increase in accuracy that derives from inspecting (and correcting if needed) a verbatim depends on whether the verbatim is a true positive, a false positive, a false negative, or a true negative; as a consequence, we will take $\Omega = \{tp, fp, fn, tn\}$. For instance, when evaluating $U(d_j, \Omega)$, the expression $P(tp)$ will mean "the probability that d_j is a true positive"[7].

4.1.3. *Probabilities of Occurrence*

Concerning Step 3, we need to describe how to compute $P(tp)$, $P(fp)$, $P(fn)$, and $P(tn)$ for each verbatim d_j.

First of all let us note that, if the verbatim has been assigned the code, then $P(fn) = P(tn) = 0$, so we are left with computing $P(tp)$ and $P(fp)$, i.e., the probability that the verbatim has been coded correctly and the probability that the verbatim has been coded incorrectly, respectively; however, $P(fp) = 1 - P(tp)$, so we only need to compute $P(tp)$. Similarly, if the verbatim has *not* been assigned the code, then $P(tp) = P(fp) = 0$, so we are left with computing $P(tn)$ and $P(fn) = 1 - P(tn)$, i.e., the probability that the verbatim has been coded correctly and the probability that the verbatim has been coded incorrectly, respectively.

So, for each verbatim, the only thing we need to do is to compute the probability that the verbatim has been coded correctly (let us denote it as $P(cor)$). Our automated verbatim coding system helps us in this, since whenever it automatically codes a verbatim it returns, along with a binary decision ("the code is assigned" or "the code is not assigned"), a numerical score of confidence in its own decision; the higher the score, the higher the confidence[8]. If we trust our classifier we can take this confidence score as a proxy for $P(cor)$; that is, we assume that the more confident the classifier is in its own decision, the higher the probability that the verbatim has been correctly coded.

to give a gentle introduction to the main intuitions of that approach while at the same time abstracting away from the hard-core maths of the original paper. Additionally, in this paper we present new experimental results obtained on survey coding datasets, while the experiments presented in (Berardi et al., 2012) had been run on datasets of newswire reports.

[7] Note the difference in notation: by TP (uppercase letters) we indicate the number of true positives as deriving from the classification of a given set of verbatims, while by tp (lowercase letters) we indicate the event of being a true positive, as in "the probability that the verbatim being inspected is a true positive".

[8] This is not a prerogative unique to our system; most modern classifiers, be them based on machine learning or not, return such a confidence score.

Technically, a confidence score is not yet a probability, since confidence scores (at least: those output by our system) can take values from 0 (lowest) to $+\infty$ (highest), which are not probabilities since they do not range on [0,1]. Luckily enough, we can convert confidence scores into probabilities via *probability calibration* methods. Discussing them is beyond the scope of this paper, so we refer the interested (and mathematically well-equipped) reader to (Niculescu-Mizil and Caruana, 2005) for a general discussion of probability calibration, and to (Berardi et al., 2012) for a discussion on how probability calibration is in fact realized in our method.

For our purposes, we may now simply take for granted that the probabilities discussed in Step 3 can indeed be computed.

4.1.4. *Gains*

Concerning Step 4, it is natural to define gain $G(\alpha_j, \omega_i)$ as the increase in accuracy that derives from inspecting and correcting verbatim d_j if event ω_i occurs. For instance, $G(\alpha_j, fp)$ will be defined as the increase in accuracy that derives from inspecting and correcting verbatim d_j if it turns out to be a false positive.

First of all, let us note that $G(\alpha_j, tp) = G(\alpha_j, tn) = 0$, since there is no increase in accuracy that derives from inspecting a verbatim that is already coded correctly. So, only $G(\alpha_j, fp)$ and $G(\alpha_j, fn)$ are of interest here.

$G(\alpha_j, fp)$ may naturally be defined as the increase in F_1 that derives from correcting a false positive, i.e., by removing a false positive from and adding a true negative to the contingency table:

$$G(\alpha_j, fp) = \frac{2TP}{2TP + (FP - 1) + FN} - \frac{2TP}{2TP + FP + FN} \tag{3}$$

where the first summand is F_1 after the correction and the second is F_1 before the correction. Analogously, $G(\alpha_j, fn)$ may be defined as the increase in F_1 that derives from correcting a false negative, i.e., by removing a false negative and adding a true positive:

$$G(\alpha_j, fn) = \frac{2(TP + 1)}{2(TP + 1) + FP + (FN - 1)} - \frac{2TP}{2TP + FP + FN} \tag{4}$$

However, the problem in this formulation is that the quantities TP, FP and FN are not known!, since at the time of computing the ranking we do not know if a verbatim has been coded correctly or not. Luckily enough, if we do not know these quantities we can at least *estimate* them. This can be done via a well-known technique called 10-*fold cross-validation* (see also Figure 3), which essentially consists of partitioning the training set into 10 equally-sized subsets of verbatims, and running 10 train-and-test experiments, each of which consists of using one of the 10 subsets (always a different one) as the test set and the union of the other 9 as the training set. The final contingency table is formed by taking the 10 contingency tables generated by the 10 experiments and pooling them together into a single one (so that, e.g., TP contains the sum of the 10 TP's from the 10 experiments). If this final contingency table tells us that, say, 27% of the training verbatims are true positives, we make the assumption that 27% of the *test* verbatims are true positives too; this process gives us an estimate of the quantities TP, FP, FN, needed for estimating $G(\alpha_j, fp)$ and $G(\alpha_j, fn)$ as from Equations 3 and 4.

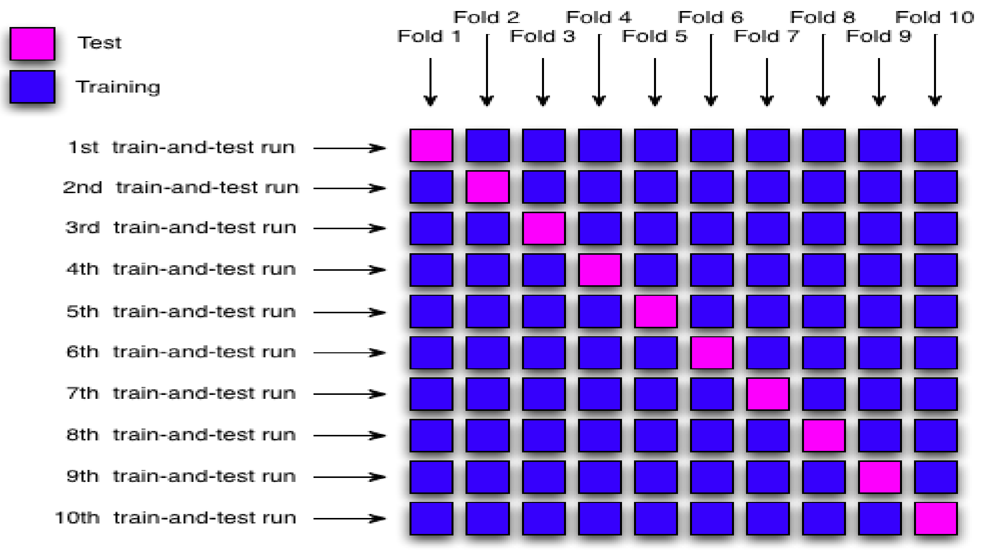

Figure 3. A graphical depiction of the 10-fold cross-validation process. Each row represents an experiment ("run") in which the 10 equally-sized parts ("folds") into which the dataset has been partitioned, are subdivided between training data (the 9 blue boxes in a given row) and test data (the magenta box in the same row).

We have now reached the point at which the gains discussed in Step 4 can indeed be estimated. This completes the description of our utility-theoretic method for ranking the automatically coded verbatims.

5. Evaluating the Effectiveness of Ranking Methods

What we need now is a methodology for testing the effectiveness of our utility-theoretic ranking method over real-world datasets and for comparing it with the effectiveness delivered by other ranking methods. We will start describing this methodology by introducing a suitable notion of "error reduction" (i.e., increase in accuracy).

Given that we have employed F_1 as a measure of accuracy, the function $E_1 = (1 - F_1)$ is a suitable measure of error. Let us define *error at rank* k (noted as $E(k)$) as the amount of E_1 still present in the set $D = \{d_1, ..., d_n\}$ of automatically coded verbatims after the coder has inspected the verbatims at the first k rank positions. Given this definition, $E(0)$ is simply the initial error generated by the automated classifier (i.e., the value of E_1 before any verbatim in D has been inspected), and $E(|D|)$ is 0 (where $|D|$ indicates, as customary, the number of elements in D). The notion of "error at rank k" allows us to define our final evaluation metric, *error reduction at depth* x (noted as $ER(x)$), defined as the reduction in E_1 obtained by the human coder who inspects the verbatims at the first $x \cdot |D|$ rank positions, i.e.,

$$ER(x) = \frac{E(0) - E(x \cdot |D|)}{E(0)} \tag{5}$$

For instance, $ER(0.25)$ is the reduction in E_1 obtained by the human coder who inspects 25% of the verbatims in D, starting at the top and working down the ranked list. It is clear from the definition that $ER(x)$ ranges on the $[0, 1]$ interval, where 0 indicates no reduction at all and 1 indicates total elimination of error.

Figure 4 illustrates the notion of error reduction graphically. The x axis represents the *inspection depth*, i.e., the fraction of the set D that the human coder has inspected, while the y axis represents the error reduction $ER(x)$ achieved for a given such depth. The four coloured curves indicate, each for a different ranking of the same set D, the error reduction obtained by a human coder who inspects the verbatims in D in the prescribed order. For instance, a human coder who inspects only 20% of the verbatims in D will obtain a reduction in the overall error present in D of about 0.20 if the ranking has been performed by the "blue" method, and of about 0.40, 0.60, and 1.00, if the "green", "red", and "pink" methods, respectively, have been used instead. So, higher plots represent more cost-effective methods, i.e., methods that bring about higher levels of increase in accuracy (or reduction in error) for the same amount of human coder inspection effort. All the curves start at the origin of the axes (meaning that, no matter what ranking method is used, no reduction in error is obtained with no inspection effort) and end in the upper right corner (meaning that, no matter what ranking method is used, all error is eliminated if the human coder inspects all verbatims in D).

6. Experiments on Real-World Datasets

Since we now have a measure of the effectiveness of a given ranking method, we are ready to specify the experimental protocol we are going to use. Given a dataset consisting of verbatims manually coded according to a code c, we go through the following steps[9]:

1. We split the dataset into a set of training verbatims and a set of test verbatims; we train a classifier from the training verbatims;

[9] This experimental protocol, and the fact that in its description we call the codes manually assigned to the verbatims their "true codes", might be taken to imply that human coders are reliable and consistent, which we know is usually not the case in practice. However, this is actually a simplifying assumption that is *necessary* for making the experiments themselves possible. All the experiments aimed at assessing the accuracy of automatic coding systems (not only the ones discussed in this paper) need to assume that the test set has been coded by an authoritative coder, in the sense that they need a "gold standard" against which the results of the automatic system should be checked; no gold standard, no accuracy tests. If the test set being used for the experiments has instead been coded unreliably and inconsistently, this simply means that the accuracy values computed on it will be a pessimistic estimate of the accuracy that would be obtained if the test set had been coded reliably and consistently.

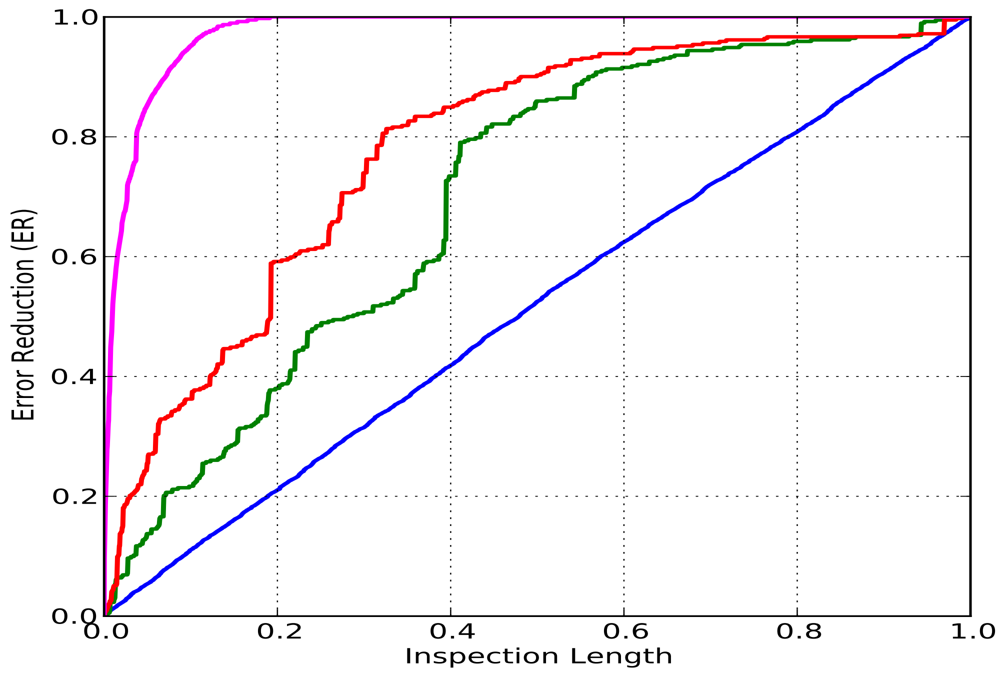

Figure 4. Four curves representing four levels of error reduction as a function of the inspection depth, as deriving from four different ways of ranking the verbatims in D.

2. Using the classifier trained in Step 1, after hiding their true codes to the classifier we recode all the test verbatims; for each such verbatim, the classifier returns a binary decision and a confidence score; we fill the contingency table by checking the binary decisions returned by the classifier against the corresponding true codes;

3. Among the test verbatims not yet inspected, we find the one that maximises expected utility, as detailed in Section 4.1;

4. We check the code assigned to this verbatim by the classifier against its true code; if they are the same we mark it as inspected and go back to Step 3, otherwise

   a) We correct the miscoded verbatim and mark it as inspected;

   b) We update the contingency table accordingly and compute $ER(x)$ as from Equation 5;

   c) We recompute the gains as from Equations 3 and 4, and go back to Step 3.

This process simulates the activity of a human coder who, after a set of uncoded verbatims have been automatically coded, inspects all of them in the order suggested by the provided ranking. By recording all the $ER(x)$ values obtained in the process, as from Step 4b, we can generate a plot similar to the one of

Figure 4 that graphically represents the effectiveness of our ranking method. By modifying Steps 3 and 4 suitably we can generate plots corresponding to other ranking methods, so as to be able to compare their effectiveness against that of our own method.

We have subjected our ranking method to thorough experimentation on datasets of real survey data. For this purpose we have used the same datasets used in (Esuli and Sebastiani, 2010); while in that paper we tested the ability of our software system at coding verbatims automatically, we here test the ability of our ranking method at maximising the cost-effectiveness of the human coders' inspection work.

Table I lists the main characteristics of the 15 datasets we have used. 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 continuous ("tracking") survey that Language Logic LLC codes 12 times a year, which consists of "semi-open" brand questions (i.e., questions – such as "What is your favourite soft drink?" – that, although in principle eliciting a textual response, usually generate many responses consisting of only the name of a product or brand, with this name coming from a small set of such names). The next 4 datasets 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?". Actually, the Egg-A1 and Egg-A2 datasets contain the same verbatims, but the test verbatims differ in the codes applied to them, since they were coded independently by two different human coders so as to provide data for an intercoder agreement study (see e.g., (Carey et al., 2006)); so, we treat them as separate datasets. The same goes for the Egg-B1 and Egg-B2 datasets. 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. For all these 15 datasets, see (Esuli and Sebastiani, 2010, Table I) for more details[10].

[10] Since some of these datasets are fairly small (e.g., about 200 verbatims), it might be legitimate to wonder how large a dataset needs to be in order for the conclusions drawn from an experiment to be reliable. We do not address this point extensively since it would be beyond the scope of this paper. Suffice it to say that what constitutes a "statistically representative sample" of the set of verbatims that will be encountered in one's operational environment is hard to characterize in a few words, and is the main subject of the theory of sampling (see e.g., (Cochran, 1977)). For our needs we may simply stick to the (simplistic and intuitive) rule of thumb that (a) the bigger the test set, the more reliable the results we obtain on it are

Table I. Characteristics of the ten market research datasets (LL-A to LL-L), four customer satisfaction datasets (Egg-A1 to Egg-B2), and one social science dataset (ANES L/D), that we have used here for experimentation. The columns represent the name of the dataset, the number of verbatims in the dataset ($\#V$), the number of training verbatims ($\#Tr$) and the number of test verbatims ($\#Te$), the number of codes in the codeframe ($\#C$), the average number of positive training verbatims per code (AVC), the average training verbatim length (AVL), and the F_1 obtained by the automated classifier (F_1). Note that the F_1 values reported are different from those reported in (Esuli and Sebastiani, 2010) since these latter were obtained with a different experimental protocol (10-fold cross-validation) than the one used here (train-and-test).

Dataset	$\#V$	$\#Tr$	$\#Te$	$\#C$	AVC	AVL	F_1
LL-A	201	140	61	17	15.35	1.21	0.97
LL-B	501	350	151	31	20.48	1.62	0.91
LL-C	201	140	61	17	8.24	1.59	0.98
LL-D	501	350	151	27	31.30	2.05	0.84
LL-E	201	140	61	36	6.53	2.59	0.86
LL-F	501	350	151	56	26.45	3.96	0.81
LL-G	501	350	151	100	15.68	3.87	0.76
LL-H	501	350	151	84	21.73	4.83	0.73
LL-I	501	350	151	67	23.81	4.60	0.76
LL-L	501	350	151	65	20.55	3.15	0.75
Egg-A1	1000	700	300	16	86.56	1.98	0.61
Egg-A2	1000	700	300	16	86.56	1.98	0.59
Egg-B1	926	653	273	21	50.38	1.62	0.54
Egg-B2	926	653	273	21	50.38	1.62	0.52
ANES L/D	2665	1865	800	1	969.00	0.52	0.86

Figure 5 reports the results (in terms of $ER(x)$) of our experiments with five different ranking methods, each represented by a different curve in a different colour. Since we have 15 datasets, and since the codeframes used in the different datasets contain up to 100 different codes (see 3rd column of Table I), it would have been too cumbersome to report results for each individual code. As a result, for each dataset we have computed the average performance of the method across all the codes in the respective codeframe, and we have then computed the average performance of the method across the 15 datasets. So, each curve

going to be, provided that (b) the test set is an "as-random-as-possible" sample of the set of items that need to be coded.

is the result of testing a ranking method individually on a total of 594 codes[11]. Let us discuss these ranking methods.

The blue curve (marked “Random”) represents the expected $ER(x)$ values of the “random ranker”, i.e., of an algorithm that presents the verbatims in random order for the human coder to inspect. The curve is a perfect diagonal line, and shows that, e.g., by inspecting 20% of the verbatims the human coder may expect to reduce the error present in the dataset by exactly 20%. This is indeed what might be expected: we cannot expect better-than-random performance if we act randomly.

The red curve (marked “U-Theoretic”) represents our method as discussed in Section 4.1. We can see that the improvement in performance with respect to the blue curve is no less than dramatic; for instance, by inspecting 20% of the verbatims the human coder obtains a reduction in overall error of 60%, instead of 20% as obtained with the random ranker. *This clearly indicates that supporting the work of human coders via appropriate ranking methods is of fundamental importance for maximising the cost-effectiveness of their work.*

The green curve (marked “Baseline”) represents a variant of our method that, instead of using utility theory, uses probability theory alone. That is, this curve was obtained by setting the $G(\alpha_j, fp)$ and $G(\alpha_j, fn)$ values of Equations 3 and 4 to 1, and keeping them at this value for the entire process. In other words, this method only takes into account the probability of misclassification of the verbatim, and does not take into account the fact that correcting a false positive has a different impact on F_1 than correcting a false negative. We can observe that this method obtains a substantially inferior performance than the method based on utility theory, which confirms the quality of the intuitions underlying the latter.

The brown curve (marked “Oracle1”) actually represents an “idealised” (rather than a “real”) ranking method, since it represents how our utility-theoretic method would behave if we could perfectly guess the actual values of TP, FP, FN, from which the gains of Equations 3 and 4 are computed. As we noted in Section 4.1.4, since at the time of computing the rankings we do not know these values, we need to estimate them via 10-fold cross-validation. We obtained the results represented by the Oracle1 curve by “peeking” at the real values of TP, FP, FN, and feeding them to our utility-theoretic method. The reason why we engaged in this seemingly futile exercise was precisely to know how much we lose because of our (inherent) inability to perfectly estimate the actual values of TP, FP, FN. The answer is that we do lose something but this loss is not dramatic, as can be can seen from the difference between the brown and the red curves. This suggests that the estimation method we have used is reasonable.

The black curve (marked “Oracle2”) represents an even more idealised method, since it represents how our utility-theoretic method would behave if we could

[11] Unfortunately we are not allowed to display verbatim examples of the actual data we have used, due to the fact that, for all of the 15 data sets, we have obtained permission from the owners (a) to use the data set for experimentation purposes and (b) to report aggregate numerical performance data in publications, but not (c) to display any actual verbatim in publications.

perfectly guess whether a verbatim has been coded correctly or not. That is, while Oracle1 has no need to estimate TP, FP, FN, since it "knows" their true values, Oracle2 has no need to estimate the probabilities of misclassification described in Section 4.1.3, since for each verbatim it "knows" whether the verbatim has been miscoded or not[12]. Again, the reason we engaged in this exercise was precisely to know how much we lose by our inability to precisely know whether the verbatim has been miscoded or not. The answer is that we lose a real lot, as can be seen from the large distance between the black and the red curve. However, this is somehow inherent, since if we had a way to precisely know whether a given verbatim has been miscoded or not, we would have a way to correct its code assignment with no human intervention, and we would have no need of engaging human coders in an inspection task. The only way we can hope to partially close the gap between the black and the red line is by devising automatic coding systems with better "introspective capabilities" (i.e., such that the confidence scores they return are more reliable than the ones we have used as input), and by devising better probability calibration methods.

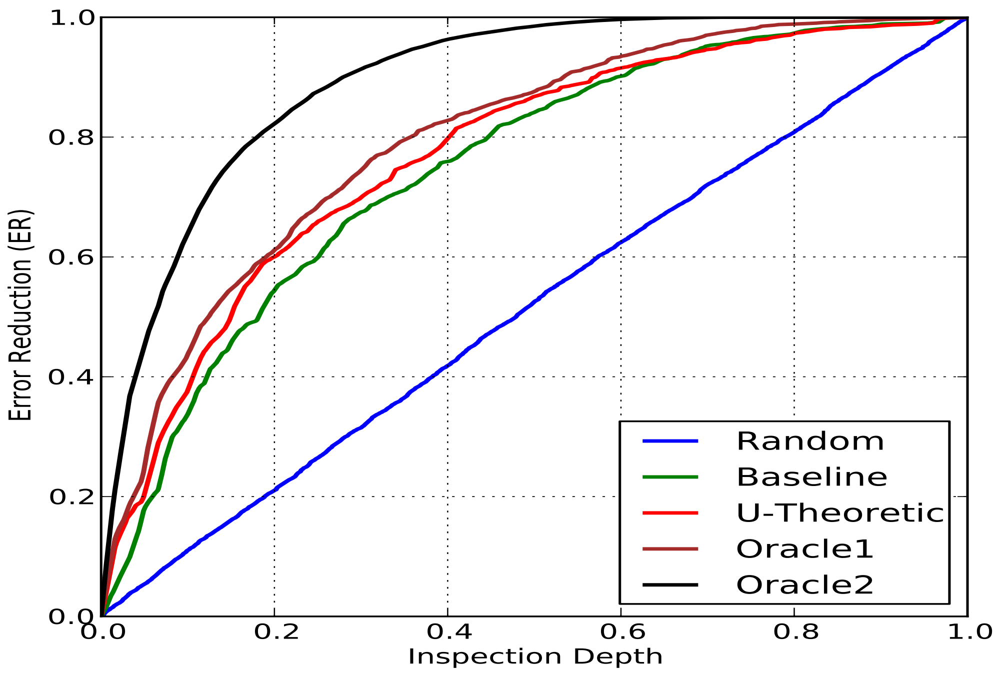

Figure 5. Five different ranking methods at work on our 15 datasets.

[12] Note that Oracle2 *a fortiori* also knows the actual values of TP, FP, FN, so it is *strictly* a more idealised method than Oracle1. Actually, it can be proven that Oracle2 is the *best possible ranking method*, i.e., it is an absolute upper bound that no other ranking method, real or idealised, can possibly outperform.

6.1. *ENER*(y)

While $ER(x)$ curves provide a nice graphical way of understanding the performance of a given ranking method, it may also be useful to have a measure that summarizes this performance in a single number. One obvious candidate might be the area under the curve that represents the method. However, one problem with this notion is that a very large portion of this area (namely, the portion below the blue curve) represents an improvement in accuracy that is due to chance (since the blue curve represents the expected performance of the random ranker); this large area tends to dwarf any difference between curves representing genuinely engineered methods (e.g., Baseline and U-Theoretic). A second problem is that this notion does not take into account the fact that lower values of x (e.g., 0.20) are more important than higher ones (e.g., 0.80), since human coders are more likely to inspect the top-ranked verbatims than the bottom-ranked ones; so, an improvement in the top 20% portion of the graph is more important than an improvement in the bottom 20%.

In (Berardi et al., 2012) we introduce a measure called $ENER(y)$ (standing for *expected normalized error reduction*). $ENER(y)$ is essentially the area between the curve representing the method and the "Random" curve, aside from the fact that it pays more importance to small values of x than to high ones. Exactly how much more importance it pays to the former than to the latter is determined by the y parameter, which indicates the expected value of the fraction of the set D that the coder is going to inspect. For example, $y = 0.05$ represents the scenario in which coders tend to inspect very small portions of the list, while $y = 0.20$ envisages coders with higher perseverance. The smaller the value of y, the less the high values of x weigh in the computation of $ENER$. As usual, the interested reader should consult (Berardi et al., 2012) for the mathematical details.

Table II lists the $ENER(y)$ values obtained by the methods we have tested, for three important values of y, i.e., 0.05, 0.10, 0.20. (As mentioned above, high values of y represent unlikely scenarios.) The $ENER(y)$ results reported in the table witness the quality of the intuitions that underlie our utility-theoretic method, which is seen to substantially outperform the probability-theoretic baseline for all chosen values of y.

7. Discussion

7.1. The Law of Diminishing Returns

All of the curves (aside from the one representing the "Random" method) in Figure 5 are fairly steep at the very beginning (i.e., for very low inspection percentages) and are decreasingly steep as these percentages increase, flattening out as they reach inspection depth values close to 1.

One practical effect of this is that the methods represented by these curves are characterized by a sort of "law of diminishing returns", which essentially says that the very first phases of annotation work are *extremely* cost-effective, Table II. $ENER(y)$ results as a function of the expected fraction of the ranked list that gets inspected. Improvements indicated for the utility-theoretic method are relative improvements with respect to the baseline. Oracle2 represents the theoretical upper bound for the performance of any ranking method.

Method	$ENER(0.05)$		$ENER(0.10)$		$ENER(0.20)$	
Random	0.000		0.000		0.000	
Baseline	0.109		0.182		0.240	
Utility-theoretic	**0.165**	(+51.3%)	**0.234**	(+28.5%)	**0.286**	(+19.1%)
Oracle1	0.196		0.270		0.320	
Oracle2	0.344		0.437		0.483	

the subsequent ones are “just” cost-effective, and the ones after them are less cost-effective. For instance, with reference to the example reported in Section 2, if the annotator’s target is to move up from the initial $F_1 = 0.533$ value to a value of $F_1 = 0.800$ (a 50% increase in F_1) she only needs to scan 25% of the ranked list, but if she instead wants to go up to 0.850 (a 60% increase in F_1) she needs to scan 40% of the list. So, requiring just (0.850/0.800 =) 6% more accuracy requires (40/25 =) 60% more annotation effort. Of course the example of Section 2 is artificial; however, it is realistic enough to illustrate qualitatively a “law of diminishing returns” that, as evident from the curves of Figure 5, also holds in practice. Note that it is indeed a goal of a good ranking algorithm for such a law to be in place; to witness, the “Random” ranking algorithm implements a situation of *constant* returns, and this is undesirable.

7.2. Easier Datasets and Harder Datasets

As discussed above, the plots of Figure 5 result from averaging across the 15 different datasets of Table I. However, it is apparent from the last column of Table I that the LL-A to LL-L and ANES L/D datasets appear “easier” (since they give rise to F_1 values between 0.73 and 0.98), while the Egg datasets appear “harder” (with values of F_1 between 0.52 and 0.61). We might wonder what kind of impact these two groups of datasets have on the collective results of Figure 5; in particular, we might wonder whether the datasets characterized by higher F_1 are also characterized by higher $ER(x)$ values. To investigate this, in Figures 6(a) and 6(b) we have plotted the results of the same five methods of Figure 5, separately averaged across the 11 “easier” datasets (Figure 6(a)) and across the 4 “harder” ones (Figure 6(b)).

Two insights can be obtained by looking at these two figures:

1. all the methods (aside from the “Random” one) perform better on the 11 datasets characterized by higher F_1 than on the 4 datasets characterized by lower F_1, as witnessed by the fact that the former curves are markedly more convex than the latter;

2. while the utility-theoretic method outperforms the baseline (probabilistic) method in both cases, the difference between the two is higher for the 11 datasets characterized by higher F_1 than for the 4 other ones.

The likely reason for the first fact is that, when a dataset is easy to classify (i.e., it gives rise to high F_1), the confidence scores that the classifier outputs are more reliable, i.e, they tend to correlate better with the true code assignments; and reliable confidence scores bring about speedier improvements, since the misclassified examples tend to be concentrated towards the top of the ranking more densely than they would if the confidence scores were unreliable.

The likely reason for the second fact is not that the latter 4 datasets are easier, but that the average imbalance between positive and negative examples (i.e., the ratio between the value in the AVC column and the value in the $\#V$ column of Table I) happens to be smaller for these 4 datasets than for the other 11 datasets; a smaller imbalance means that the difference in gain between correcting a false positive and correcting a false negative is smaller, which makes the utility-theoretic method more similar to the baseline (probabilistic) method.

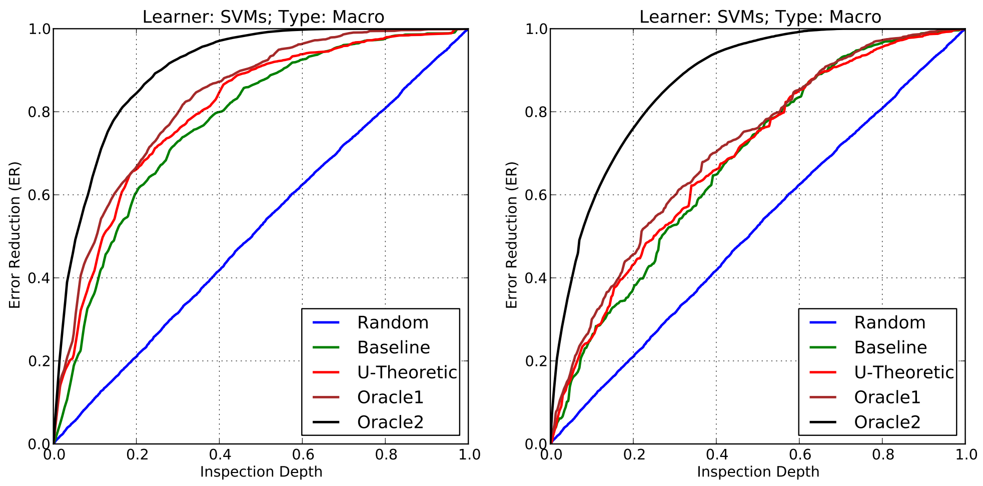

Figure 6. Five different ranking methods at work: (a) on the LL-A to LL-L and ANES L/D datasets, and (b) on the Egg datasets.

7.3. On the Practical Value of Utility-Theoretic Ranking

It is worthwhile to add a couple of observations on aspects of this method related to its practical use.

A first observation is that this method lends itself to having more than one human coder work in parallel on the same inspection task. For instance, if two human coders work in parallel, coder C' may be asked to inspect the verbatims at the odd-numbered positions in the ranking while coder C'' may be asked to inspect the ones at the even-numbered positions. In a similar vein, any number of coders may be put to work in parallel on the same task and still achieve the same cost-effectiveness guaranteed by the single-coder scenario.

A second observation is that this method is of practical value also because *it allows the human coder to have an estimate, at any stage of the inspection process, of what level of accuracy has been attained so far*. This is obtained by

1. Estimating the contingency table at the beginning of the process (e.g., after the verbatims have been automatically coded and before the inspection process begins) via 10-fold cross-validation. From the contingency table one can compute F_1, which returns an estimate of the accuracy obtained by the automated classifier.

2. Updating the contingency table after each correction is made. Again, from the contingency table one can compute an updated value of F_1, which is an estimate of the accuracy deriving from the combined action of the automated classifier and the inspection activity of the human coder.

The human coder can thus make an informed decision on when to stop inspecting, e.g., she may decide to stop when the estimate of the accuracy obtained so far coincides or exceeds the accuracy level that was requested by the customer.

8. Concluding Remarks

In more and more application contexts that require verbatim coding, the use of automated tools is a necessity, due either to the sheer amount of data that requires coding, or to strict time constraints that may be imposed on the task, or to issues of cost. However, in some of these scenarios the customer may insist that coding is to be performed according to a certain level of accuracy, a level that for some datasets may not be achievable by current automated coding technology. In these cases, the only way to achieve the goal may consist in coding the data by machine and then having one or more human coders review some of the verbatims, with the goal of performing enough corrections so as to bring accuracy up to the required level. The method we have presented in this paper explicitly attempts to maximise the cost-effectiveness of the human coders' review work by presenting the verbatims as a ranked list, with the intended meaning that the effort will be minimized if the coder inspects the verbatims in the provided order.

What are the takeaway messages that derive from this study? The first important one is that the order in which the verbatims are inspected by the human coder is of paramount importance, as witnessed by the more-than-solid improvements obtained, on 15 survey coding datasets, by two genuinely engineered ranking methods (indicated as "Baseline" and "Utility-Theoretic" in Section 6) with respect to the "Random" method (which represents a coder that inspects verbatims by picking them in random order).

The second important message is that we should be very serious about the mathematical measure we use for evaluating the accuracy of our verbatim coding systems. A first, fundamental reason (that goes beyond the specific task and methods discussed in this paper) is that the evaluation measure must reflect the specific needs of the application. For instance, if the application is such that the ability to avoid, say, false positives ("precision") is more important than the ability to avoid false negatives ("recall"), then a measure that reflects this should be chosen[13]. A second reason is that our method tailors the rankings it generates to the chosen accuracy measure (whatever this may be), since the gains of Equations 3 and 4 are defined in terms of it and are thus *optimized* for it. Altogether, these two facts point to the importance of developing a culture of evaluation for verbatim coding endeavours, be they the result of automated tools or not.

As a final (and somehow peripheral) note to the reader, we recall from Footnote 6 that the goal of this paper is to at least convey the gist of a subject matter that we elsewhere cover in much higher mathematical detail. We are indeed conscious that the technical material presented here is somehow outside the tradition of the market research literature, and may appear challenging. However, quite aside from the specific topics dealt with in this paper, we think this represents an opportunity to reflect, given the growing importance that text mining / text analytics / sentiment analysis have in market research, whether market research scholars and practitioners should or should not attempt to understand what lies beneath the surface of the text analysis tools that are being proposed to them by text analysis companies. Although this may require market researchers to upgrade their mathematical toolbox a bit, we think this should be the case, lest an entire discipline is taken hostage of vendors wanting to hide more than they want to clarify.

Acknowledgements

Most of this material was originally presented in September 2013 at the conference of the Association of Survey Computing, Winchester, UK, during a keynote talk delivered by third author, who would like to thank the Conference Chairmen David Birks and Tim Macer for inviting him to deliver this talk in such a stimulating environment. Thanks also to Ivano Luberti for many interesting discussions, and to two anonymous reviewers for providing many stimulating comments.

References

Alpaydin, E.: 2010, *Introduction to Machine Learning.* Cambridge, US: The MIT Press, 2nd edition.

Anand, P.: 1993, *Foundations of Rational Choice under Risk.* Oxford, UK: Oxford University Press.

[13] F_1 is an instance of a more general class of evaluation functions called F_β, where $\beta \in [0, +\infty)$ is a parameter that sets the relative importance of precision and recall (see (Sebastiani, 2002, page 36)). Values of β smaller than 1 emphasise precision at the expense of recall, values higher than 1 emphasise recall at the detriment of precision, while $\beta = 1$ enforces equal attention to precision and recall.

Berardi, G., A. Esuli, and F. Sebastiani: 2012, 'A Utility-Theoretic Ranking Method for Semi-Automated Text Classification'. In: *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2012).* Portland, US, pp. 961–970.

Berardi, G., A. Esuli, and F. Sebastiani: 2014, 'Utility-Theoretic Ranking for Semi-Automated Text Classification'. Technical report, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, IT. Forthcoming.

Carey, J. W., M. Morgan, and M. J. Oxtoby: 2006, 'Intercoder Agreement in Analysis of Responses to Open-Ended Interview Questions: Examples from Tuberculosis Research'. *Cultural Anthropology Methods* **8**(3), 1–5.

Cochran, W. G.: 1977, *Sampling Techniques.* New York, US: John Wiley & Sons, 3rd edition.

Esuli, A. and F. Sebastiani: 2010, 'Machines that Learn how to Code Open-Ended Survey Data'. *International Journal of Market Research* **52**(6), 775–800.

Giorgetti, D., I. Prodanof, and F. Sebastiani: 2003, 'Automatic Coding of Open-ended Questions Using Text Categorization Techniques'. In: *Proceedings of the 4th International Conference of the Association for Survey Computing (ASCIC 2003).* Warwick, UK, pp. 173–184.

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.

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 2007).* Brighton, UK.

Martinez-Alvarez, M., S. Yahyaei, and T. Roelleke: 2012, 'Semi-automatic Document classification: Exploiting Document Difficulty'. In: *Proceedings of the 34th European Conference on Information Retrieval (ECIR 2012).* Barcelona, ES.

Mohri, M., A. Rostamizadeh, and A. Talwalkar: 2012, *Foundations of Machine Learning.* Cambridge, US: The MIT Press.

Niculescu-Mizil, A. and R. Caruana: 2005, 'Obtaining Calibrated Probabilities from Boosting'. In: *Proceedings of the 21st Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI 2005).* Arlington, US, pp. 413–420.

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

von Neumann, J. and O. Morgenstern: 1944, *Theory of Games and Economic Behavior.* Princeton, US: Princeton University Press.