Machines that Learn and how to Code Open-Ended Survey Data: Underlying Principles
Fabrizio Sebastiani

Thanks Skip. We will begin by speaking about machine coding open-ended data. I will not simply present machines that do code data; I will present machines that will generate automatic coders for data based on specific {?}. I will touch on general issues of these machines. In order to exemplify this approach I will discuss one of our systems that we have – we have been designing it for the last 4 years. It is called VCS which stands for automated verbatim coding system. I will always call an open-ended answer a verbatim. I will call {?}.

I will first introduce the basic principles of which VCS works and I will {?}. I want to show you what the accuracy and the efficiency of VCS is on a set of real data. I will define accuracy and I will look at types of accuracy – accuracy of the individual level meaning the coding of the individual survey responses and the accuracy of {?} which can be defined as accurately guessing the percentage of respondents that actually are coded under your code.

Let us start. As I said we have been working with this system for the last four years. Actually there was a very initial prototype that we designed maybe 6 years ago that we ran data provided by NRC and published a paper on this. {?} at the conference of the Market Research Society last year. {?}

Originally the system was developed for a very specific customer, Egg plc. Those of you who come from the UK know Egg. It is the largest purely online bank in the world. It has now been acquired by Citigroup. Actually they still owe us some money and I am getting nervous about it. This was originally deployed in July, 2006 and is now fully operational. Egg acquired the system but they wanted us to manage the – they wanted the system to manage their customers' satisfaction surveys. They get customer satisfaction data through questionnaires over the web. They code 20,000 questionnaires/month. They also use this system to code huge backlogs of verbatim that they had never had time to code.

VCS is also available in a large platform called Ascribe by Language Logic LLC. It covers 5 major European languages (English, Spanish, French, German and Italian). You can use VCS for coding your studies in any of these languages.

What is the underlying philosophy? VCS is an adaptive system – it should be used for coding verbatim responses under any user-specified code frame. Given such a code frame, VCS automatically generates an automatic coder. We have used these for several applications including coding open-ended responses such as customer satisfaction with Egg. Several market research applications have shown some results that we provide for language logic. In social/political studies will show results of coding. Actually the basic unit along which VCS works is not the code frame as a whole, it is the code. So we break down code frame into different codes and generate a binary coder for each such code. A binary coder is something that takes a verbatim and decides whether the verbatim should or should not be an applied code. That is independent of any other binary coder. The applied concept is that it is the case that the same verbatim could be assigned 0, 1, or 7 codes for the same verbatim. You might say what happens if I want to assign one and only one code? There are several different ways that can take care of the application but the default is that 0-1 or several codes can be applied independently of each other.

VCS is based on a learning metaphor: the system learns from manually coded data the characteristics of the new verbatim should have in order to be attributed the code. We manually code the data that you need to provide the system in order to train the system should include both positive examples of the code and negative examples of the code. The system needs to learn where border between the two lies. As you all know coding is a subjective task. So does the VCS try to learn to objectively correct classifier – no? VCS only tries to replicate the subjective coding behavior of the human coder who has coded the training examples. The basic lesson is that if you want to have an accurate system, you better train the system. We use manually code examples from a reliable or expert human coder.

Providing manually coded examples of the code to the system is by no means different than providing a child, when you want to train a child to recognize a tiger, you provide images or photographs of tigers. So you might want to provide your child with a quality example, this is a tiger, this is another tiger and these are also tigers. You also provide negative examples – this is not a tiger and these are not tigers. At this point you have trained your kid to recognize tigers. You might want to check to see if he understands this concept so you might ask him if this is a tiger. This is a key to coding. At this point you might want to give your kid 100 photographs of cats and ask him are these tigers or not. Your kid might realize he is still unsure about the notion of a tiger in some cases. He might come back to you. For a select number of images of which he is uncertain, he might ask you – is this a tiger? By giving him feedback and saying yes, this is a tiger – it reinforces his notion of what a tiger is. You increase his future accuracy in recognizing a tiger.

This is the basic makeup of the system. On this slide this is basically describing the architecture of the system. I want to know focus on the yellow area. The yellow area actually corresponds to the training phase of the system. You have a human coder who takes a small sample of the verbatims and manually codes them. He feeds the coded verbatims to the trainer, the learning systems, which generates automatically the verbatim coders. The second phase is where you take the verbatim codes that you have generated and you want to apply them to coding large masses of uncoded verbatims. You would feed the verbatim coders you have generated and the uncoded verbatims to the coding engine. The coding engine would apply the codes. You find yourself with 100,000 verbatims which are coded. You can use them for several applications. One is usually customer satisfaction. Suppose one of the codes is customer might defect to the competition. {?}

We go to the coding phase in which you apply what you have learned. As I was mentioning there is a phase which we call the {?} phase. VCS might come back to you and say all the coded verbatims get fed into a ranking engine that ranks or sorts the verbatims in terms of the confidence with which the verbatims have been coded. The VCS {?}. The top most verbatim in the list would be the verbatim in which the computer is much uncertain about. It is the very first verbatim that the user is asked or suggested to validate or give feedback. Let's assume that the user has the time or the manpower to validate for 100. The human coder would take the first 100 documents of verbatims and validate them and give them back to the trainer. This is training so the system can be retrained to build new automatic classifiers and thus forth more accurate. They are going to be more accurate especially in the areas in which you gave feedback about. It is the case that the verbatims that are somehow similar to the ones the user has given feedback will be coded more accurately.

What are the advantages of using VCS? There is no need for an expert to write coding rules in arcane language so the system only needs user-coded examples for training. If you can't code at all you can operate VCS. You don't need to be any more computer safe than this.

The second advantage is that it is very easy to update the system to a revised code frame. Supposed that in the course of the operation you want to add a new code – you only need to feed new coding examples or new training examples for the new code. It is very easy to update to revise the code frame or a brand new code frame or a brand new survey – you only need to add the human coded examples.

The other advantage is that it doesn't require any domain-dependent resource; it is a plug and play system. You simply stick in the piece with your data and go and have coffee and you get your results back.

The system has pretty good accuracy at the individual level and we have come to find excellent accuracy at the aggregate level and also excellent learning and coding speed.

Let me come to the issue of accuracy. What is accuracy? By accuracy of the coding system I mean the expected frequency with which the coding decisions of the system are expected to agree with the coding decisions that the coder who has coded the training verbatims. I don't refer to the letter of agreement of its decision of an expert coder. If you have trained the system with data from a junior coder then what you can expect is to replicate the performance of a junior coder. Accuracy means how

well the system can replicate the subjective judgments of the person which it was trained by providing manually coding examples.

How do we estimate the accuracy? We estimate the accuracy by comparing the system's decisions with those of the human coder on one or more test datasets or studies. Each dataset studied consists of a set of manually coded verbatims plus the corresponding code frame.

Why do we make a distinction between individual and aggregate accuracy? We measure accuracy at the individual level as defined as the perfect system is the one which for a code C, assigns C to the verbatim exactly when the human coder has assigned C. What is the definition of the aggregate level accuracy? In the perfect system in terms of aggregate accuracy is the one which, for a code C, assigns X% of the verbatims to C exactly when the human coder would have assigned X% to the verbatims of C. Suppose that Matt Gonard asked us to run a survey on their customer satisfaction data. Suppose that the human coder coded the entire survey; I would have come to the conclusion that 46% of the McDonald's customers are happy with the Big Mac. For us the perfect system is the one that comes to the same conclusions that 46% of the customers are happy with Big Mac. Why is the aggregate level more lenient than the individual level? For the same reason that if your system makes an equal number of false positives and false negatives it is perfect on the aggregate level but it is not perfect on the individual level. So why do we measure both levels? In customer satisfaction you are interested in individuals because if you have an unhappy customer you want to call him up. Why in market research are you typically interested in the aggregate accuracy? McDonald's wants to know exactly how many/what percentage of the customers are happy with the Big Mac. We aren't interested in knowing if Joe Plumber is happy with the Big Mac.

A system that is accurate at the individual level is also accurate at the aggregate level but not vice versa. You will see in our system which is reasonably accurate at the individual level is much more accurate at the aggregate level because it is tuned to be very accurate at the aggregate level. If it is tuned to choose between two options, two different coders that have equal individual accuracy it will choose the one that bonds as the best the number {?}.

Let's discuss accuracy measures. The accuracy testing requires an accuracy measure. We actually adopted a measure called F1 which is the industry standard for text classification and coding. What is F1: It gives sort of a mean of two different notions. Precision is defined as the ability of the system to avoid over coding – excessively assigning an even code. It is the ability of the system to avoid false positives. Recall is quite opposite. It is to measure the ability of the system to avoid under coding. It measures the ability of the system to avoid false negatives. A good system is one that tends to {?} false positives and false negatives. Because of this you want to have a measure that is somehow a mean or {?}. We see that F1 is such a measure.

F1 is known as the harmonic mean of precision and recall. It can easily be computed through an experiment through a contingency table. In these two slides I have a few numbers, examples for computing F1 for Code C and for another called J. The interesting thing is of course that you want even in an experiment to have an aggregate measure of F1 across different codes. You can easily come up with the measuring by simply adding the corresponding cells into a collective which is just a number. For the example, I have here we have a .692 value of F1. We have come now to where this is good enough.

Why is F1 a good measure of accuracy? F1 it is equal to zero for the perfect system where the system has no true positives and no true negatives. It is equal to 1 for a perfect system; no mistakes, no false negatives and no false positives. It is interesting because it partially rewards partial success. If the true codes, the codes that should have been assigned, to a verbatim – C1, C2, C3, C4 well attributing C1, C2, C3 is reward more than attributing just C1.

This third property which means very much to us computer scientists, I'm not sure what it means to most of you. Measuring is not easy to gain. It means that you cannot easily set up a trivial coder and have high accuracy according to this measure. This trivial coder is a coder that always says no, it

rejects, it assigned no code. There are accuracy measures that would give good results and we don't want these. The trivial acceptor or the trivial rejecter or the random coder gets variable results from F1.

This feature I have already talked about – it rewards systems that balance precision and recall. Why? Because it is a measure that given a certain number of mistakes – if T is better so F1 is higher – if the mistakes are equally subdivided into false positives or false negatives – we can't do this. There is also the symmetric measure. The agreement between the system and coder is the same as the agreement between the coder and the system. This means that the symmetric measurement is good, you know the agreement between two different coders without making any hypothesis whether Coder A is more expert than Coder B. This has allowed us in a couple of cases to {?}. {?} which we compare the agreement of system Coder A to the agreement of system of Coder B and the agreement of Coder A with Coder B. It means that VCS has no agreement with each of these coders. It should just be even weight with respect to the agreement of the two coders. If the task is very hard it is quite certain that VCS will have {?}.

I guess I only have 5 minutes here. I will just sketch a few examples that I have here. Here we have a table with 13 different experiments that we ran. The first ones labeled LL-something are market research data sets coming from language logic. The other two, Egg-A and Egg-B are customer satisfaction data sets. The last one is ANES data. How did we experiment {?}. We had two different data sets; one for people who said yes and one for people who said no. Here the task that we set ourselves was to ignore, to pretend that we didn't know that the people had said yes or no and decide based on the verbatim whether the person had said yes or no. The last column which is highlighted in yellow is F1 which basically gives you a measure on how good we have been with these data sets. As you can see we have been reasonably good. ANES is .86 is much closer to the theoretical perfect accuracy system of 1 than to the perfect system is basically zero.

You can see the different levels of accuracy based on documents ranging from .92 LL-E to .60 Egg-B. Why is there is such a wide distinction in performance? There are different characteristics for each of these data sets. For instance here we are showing the number of training examples we had. The higher the number of training examples you can expect more accuracy.
This is another data set that we have tested. We see the size of the code frame. Another thing that is quite important is the average number of training examples per code. It is all very useful to know that we have lots of training examples if the codes are very many. What we had to do was to check how many training examples per code we must have. This is not the only barometer. This is what we might characterize as the informational coding. There are a number of {?} which basically has to do with the amount of linguistic viability in the data sets. If the data set is very varied in terms of linguistic expression well it is {?}. Just to give you a quantification of how varied the data set is – this is the size of the dictionary that has been used – the number of unique words that come up in the training set. You might see that in ANES data set this is much higher.

How do these barometers come up with a fine measure? One measure that we can extract from this is what we call the relevant amount of information which is the ratio between the average number of training examples of a code and the amount of linguistic variability of the data set which can sort of be graph indicated by the number of unique words that come up in the set. You see that there is quite a nice correlation between the quantity of information provided by the training set and F1. {?} There is a way to estimate from the training set what the level of accuracy is going to be.

I guess my time is about over but I just wanted to touch briefly on the PD measure, the idea of testing accuracy on the aggregate. If you want to test accuracy on the aggregate you don't use F1. We use this measure called PD.

This is the discrepancy between the true percentage and the predicted percentage of verbatims we code C. Suppose that 2% is used 46% and suppose that VCS gets it wrong, it says 43%. So PD is equal to 3%. The perfect system has PD=0. We have imported a few values of PD to get an idea how useful the system would be if you are only interested in accuracy of the aggregate. For each of these data sets we list the maximum discrepancy across the codes or the average discrepancy across the

codes. In the yellow column, LL-A, you see that on average across the 18 different codes in the code frame, the average discrepancy was .008 so it was less than 1%. This is pretty good. We didn't perform very well on ANES. This is still something that I haven't really understood why. We preformed very well on the individual but we didn't perform very well on the average. On average we got it wrong .046 which means in a sense that group percentage was 52% so instead we said 57%. The pink column refers to the maximum discrepancies across the different codes. It refers to the code on which you performed the {?}.

I want to finish this with a graphical illustration of what these mean. For instance one of these data sets LL-E, the blue bar represents the percentage that should have been predicted while the red bar illustrates the percentage that hasn't been predicted by the system. You can see that for code 354, we didn't do very well. We predicted 7.5% instead of 10%. For the other codes we did reasonably well. I will skip the rest of the presentation because I want to leave space for questions. My slides will be available to all of you so you can look at the rest of the presentation.