nCoder+: A Semantic Tool for Improving Recall of nCoder Coding

Zhiqiang Cai[1](✉), Amanda Siebert-Evenstone[2], Brendan Eagan[2], David Williamson Shaffer[2], Xiangen Hu[1,3], and Arthur C. Graesser[1]

[1] The University of Memphis, Memphis, TN 38152, USA
zhiqiang.cai@gmail.com
[2] University of Wisconsin-Madison, Madison, WI 53706, USA
[3] China Central University, Wuhan, Hubei, China

Abstract. Coding is a process of assigning meaning to a given piece of evidence. Evidence may be found in a variety of data types, including documents, research interviews, posts from social media, conversations from learning platforms, or any source of data that may provide insights for the questions under qualitative study. In this study, we focus on text data and consider coding as a process of identifying words or phrases and categorizing them into codes to facilitate data analysis. There are a number of different approaches to generating qualitative codes, such as grounded coding, a priori coding, or using both in an iterative process. However, both qualitative and quantitative analysts face the same coding problem: when the data size is large, manually coding becomes impractical. nCoder is a tool that helps researchers to discover and code key concepts in text data with minimum human judgements. Once reliability and validity are established, nCoder automatically applies the coding scheme to the dataset. However, for concepts that occur infrequently, even with an acceptable reliability, the classifier may still result in too many false negatives. This paper explores these problems within the current nCoder and proposes adding a semantic component to the nCoder. A tool called "nCoder+" is presented with real data to demonstrate the usefulness of the semantic component. The possible ways of integrating this component and other natural language processing techniques into nCoder are discussed.

Keywords: Coding · Grounded coding · A priori coding · Automatic coding · Grounded theory · Qualitative analysis · Quantitative analysis · Latent Semantic Analysis · Topic modeling · Machine learning

1 Introduction

When researchers analyze text data, they often search for culturally relevant and meaningful aspects of a discourse. For example, people that are interested in understanding how students think during science curriculum might look for science content knowledge (e.g. nitrogen cycle knowledge) and for scientific practices (e.g. developing and using models). However, in order to make the claim that a student exhibits scientific knowledge or practices, researchers need to provide evidence for their interpretation. One way to find evidence in data—and eventually qualitative interpretations—is to develop a set of codes that allow researchers to systematically categorize phenomena in their data to help identify patterns [1, 2].

However, in the age of digital learning environments and ubiquitous data collection, we have more information than ever about what students are doing and how they are thinking. The sheer volume of this data can render traditional qualitative methods unfeasible. One way to address this issue is to create automated codes that apply some set of rules to a dataset to assign values to each piece of data.

To aid researchers, Shaffer and his colleagues created a system for scaffolding automated classifier development. The nCoder is a learning analytics platform used to develop, refine, validate, and implement automated coding schemes. The nCoder is designed specifically for working with large and small sets of text data, such as interviews, transcripts, logfiles, and other text data. Users can generate codes by defining a construct, identifying common words associated with the construct, testing their code, and then updating their construct definition or wordlist until the researcher achieves an acceptable level of agreement between their coding and the automated coding scheme.

In this paper, we first give an in-depth description and analysis of the nCoder coding process. Then we discuss a particular problem with *recall* during this process. Recall is the ratio of the number of items coded by a classifier to the total number of items that should be coded. In nCoder, it is possible to have a situation, when the frequency of a code is low, for the recall to be low, even if the kappa is high enough to achieve a statistically significant rho. That is, it is possible to achieve acceptable agreement and generalize that agreement to the dataset yet have a potentially high rate of false negatives. After identifying and explaining this problem, we describe a potential solution to this problem – a newly developed tool equipped with a semantic component that helps solving the low recall problem. This tool is called “nCoder+”, which implies that this tool is simply an add-on to nCoder. A real data set will be used to illustrate the usefulness of this tool. We will end the paper with discussions on integrating this new component and other possible techniques into nCoder.

2 Approaches to Coding

Coding is a process of assigning meaning to a given piece of evidence. Evidence may be found in a variety of data types, including documents, research interviews, posts from social media, conversations from learning platforms, or any source of data that may provide insights for the questions under qualitative study. In this study, we focus on text data and consider coding as a process of identifying words or phrases and categorizing them into codes to facilitate data analysis.

There are a number of different approaches to generating qualitative codes, such as grounded coding, a priori coding, or using both in an iterative process [3]. Grounded coding, also referred to as inductive, emergent, or bottom-up coding, is an exploratory process which allows a researcher to discover new concepts and theories that emerge from the text data [4]. Researchers are encouraged to read through the data line by line and identify concepts that may construct in-depth understanding of the data [4, 5]. One major challenge in grounded coding is generating new concepts. Since it is exploratory, a researcher may iteratively refine the definition of the concepts, which implies data re-coding. When the data size becomes too large, grounded coding by hand can become time-consuming if not impractical.

A priori coding is another identification process, in which a coder identifies pre-defined concepts from an existing theoretical framework. A priori coding, also referred to as theoretical, deductive, or top-down coding, starts with a theory or set of constructs and then searches for the ideas in the data rather than using the ideas in the data. In our example above, someone interested in identifying science practices from the Next Generation Science Standards may search for qualitative evidence of students using one or multiple of the eight science and engineering practices.

Whether a researcher starts with the data and creates categories or starts with categories and identifies them in the data, researchers face challenges of coding reliably and consistently. First, researchers often work to validate codes in their data to ensure a common understanding of concepts between two raters[1]. Inter-rater reliability (IRR) measures assess whether two raters assign codes consistently in the same way. To determine whether two raters have identified the same properties in the data, researchers may use tests of agreement, such as Cohen's kappa, to quantify to what degree the two raters agree with each other. A common heuristic in studies of CSCL (Computer Supported Collaborative Learning) is for researchers to sample 10–20% of their data to assess IRR [6]. Again, when the data size is large, manually coding this much data can become difficult if not impossible, especially if multiple IRR analyses need to be performed to achieve acceptable reliability between raters.

Advances in computer and natural language processing technologies have made another coding approach more accessible to researchers: *automated coding*. An automated coder or classifier is an algorithmic process that identifies whether a piece of data belongs in a certain category or class. For example, topic modeling is capable of automatically finding latent topics in a text data set and "code" the data by topic proportion scores [7, 8]. While this approach is automated and requires no human coding, researchers may find that not all topics can be easily interpreted and verified.

An ideal computer coding tool does not have to be fully automated. Instead, it should have enough flexibility to allow researchers to discover concepts they think important. Once some concepts are discovered, the tool should be able to "learn" from a relatively small sample of human coded data and code the data in a way close enough to the human coder. nCoder is such a tool. Before diving deep into nCoder, we first briefly introduce the concept of "codebooks", which plays an important role in the coding process.

[1] Not all researchers perform IRR tests. For example, researchers may use *social moderation*, where two or more raters code all of the data and resolve differences until they all agree on the code (Herrenkohl and Cornelius) [14].

3 Codebooks

During coding, researchers often create a *codebook* to organize and summarize information about codes. Code books are often used to communicate ideas for IRR and are often reported in the methods sections of resulting publications. Within a codebook, the name, the definition, and examples of the code are included. For example, in analyzing text data from presidential primary debates, environmental issues could be an interesting code. In a codebook, this code may look like the one shown in Table 1.

Table 1. Code book for environmental issues

Name	Definition	Example	Classifier	IRR
Environmental Issues	Referring to harmful effects of human activity on the biophysical environment	"Governor Pataki, you've indicated you believe climate change is real and caused at least in part by human activity"	\benvironment ,\bclimate ,\bpollut ,\bgreenhouse ,\bdegradation ,\bglobal warm ,\bhabitat ,\bextinction ,\bsuperfund ,\btoxic ,\bconservation ,\bsustainability ,\brunoff ,\bnatural resource	$\kappa = 1.00$ $rho = 0.01$ $n = 40$

If the researchers used automated classifiers, then the word lists, patterns, and/or rules that were applied to the data to assign the code may also be included in the codebook.

If the researchers choose to validate their codes, another section of the codebook is the results of their IRR analyses. In our case, we have added our classifier list and IRR between the human rater and classifier. In a full analysis, we would also validate our code between two human raters as well as between the second human rater and the classifier. In this analysis, we focus on this first iteration in the coding process.

The nCoder tool helps researchers build and validate their codebooks. A critical component of any statistical analysis of text data is some form of coding scheme that identifies key concepts and clusters of terms in the data. To generate valid insights, however, this coding process has to be compared to the work of human raters. nCoder minimizes the amount of data that needs to be hand-coded by using cutting-edge statistical techniques to establish the reliability and validity of codes. Once codes are validated, nCoder automatically applies a coding scheme to larger datasets quickly and efficiently, even coding new data as it becomes available. In the next section we describe how the nCoder scaffolds codebook creation and helps researchers perform code validation processes.

4 nCoder

nCoder is available as a free online tool (https://app.n-coder.org/) and an R package (nCodeR) [9] that helps researchers to code large amounts of text data by supporting the development, refinement, validation and application of automated coding schemes. nCoder also employs Shaffer's rho in reliability analyses, which again is available online (https://app.calcrho.org) as an R package (rhoR) [9, 10]. Both the R packages and webkits have been used to analyze large-scale datasets of many kinds, including chat, email, online actions, surgical performance, and brain scan data [9–14].

For this paper, we focus on code generation which is summarized in Fig. 1 as a flow chart for developing automated codes. To create a new coding project, users upload their data as either a CSV (Comma Separated Value) or Microsoft excel file. The data file may contain any number of columns, with the first row containing column names. After the file is loaded, the user is asked to specify the text column, which is used as the text data for coding. The detailed coding steps are described below.

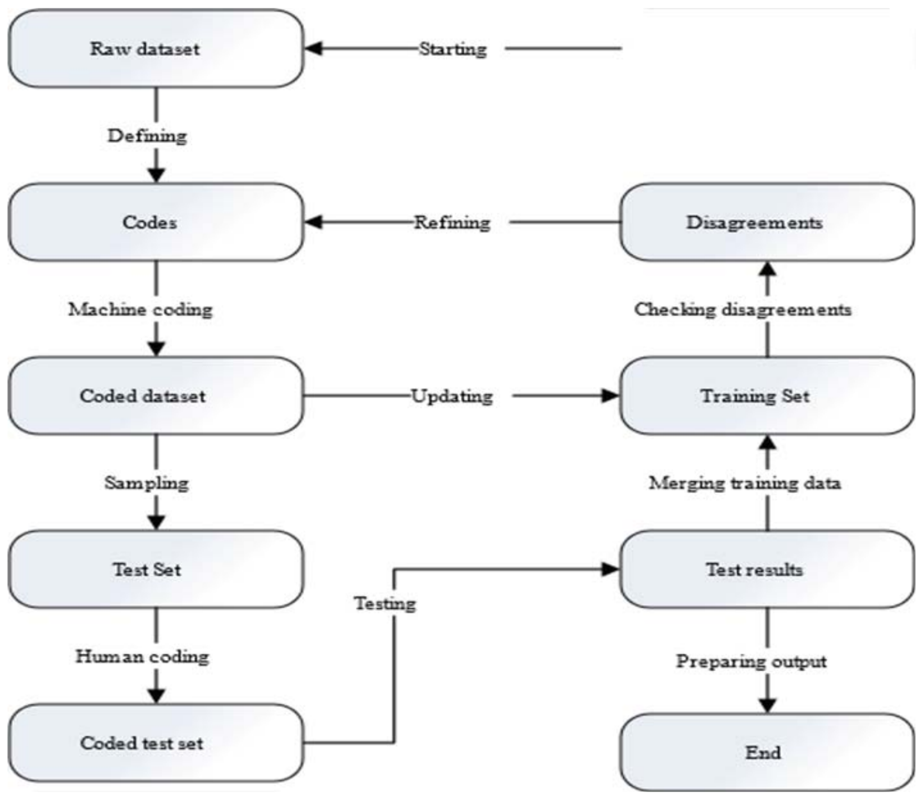


Fig. 1. nCoder flowchart

Defining Codes. The first step in coding is defining a code. In nCoder, a code definition contains three elements: a name, a description, and a list of regular expressions (RegExs). The name is the identifier of the code, which should therefore be unique. The name should also be short and meaningful so that when it is written in an article, people can easily understand and remember it. The description explains what the code is about. The description should also be clear, brief and accurate. The RegEx list specifies language patterns that can be used to code the data. For example, the RegEx "\benvironment" means that if there is a word starting with "environment" in a text cell, the code is considered present. Regular expressions are powerful in representing language patterns. nCoder provides an interface to designate word boundaries "\b". nCoder also helps users create complex combinations of words. For example, a user may want to find instances of "clean" but not "clean water". The user would search for a regular expression that begins with clean and does not include water, which is "^(?:(?!\bwater).)*\bclean(?!.*\bwater)". Such an expression may be easy for a more expert programmer and nCoder helps novice users create more nuance expressions. Users add words or expressions until they are ready to test their classifier. nCoder does not allow the user to freely go through the data because in a later step, the tool needs to sample "fresh" data for testing purpose. This restriction seems contradictory to the grounded theory method, which is built on the assumption that the concepts are discovered from navigating the data. The current version of nCoder does not allow users to do grounded coding within the tool and this feature may be built into subsequent releases. This may not be really a problem of nCoder, if we assume that the user has another offline tool (e.g., excel) to navigate through the data.

Machine Coding. In this step, the dataset is automatically coded based on the regular expressions provided by the user. nCoder uses a simple regular expression matching algorithm to automatically code the text data. It goes through each row of the specified text column, and checks if the text in the cell matches one of the regular expressions in a defined code. If yes, a "1" is assigned to the data. Otherwise, a "0" is assigned.

Test Set Sampling. A test set for each concept is randomly sampled from the coded dataset, where the minimum sample size is 10. The user may repeatedly increase sample size by 10 until the user is satisfied with the size of the test set. If a researcher wants to establish a kappa over a threshold of 0.65 they should use a test set size of at least 40. If they want to use a threshold of 0.9 they should use a test set size of at least 80. nCoder provides a measure "rho" to indicate whether or not the sample size is large enough. A rho higher than 0.05 when the kappa is higher than a threshold of interest in the test set, indicates that the test set size should be larger. When a targeted concept has a low occurrence in the dataset (e.g., <20%), a small test set may not be able to represent the targeted concept and can cause difficulties in achieving sufficient IRR. nCoder uses a technique, called "inflation" to solve this problem. That is, when creating test sets, nCoder guaranties at least 20% of the lines in the test set contain the code. For example, for a test set size of 10, nCoder would randomly pick 2 lines that the algorithm coded positively and then randomly select the final 8 lines from the rest of the data.

Human Coding. After the test set is generated, the sample texts are presented to the user for coding. The user selects a concept to code and the corresponding test items are displayed, together with a "Yes" button and a "No" button. The user codes the test items by pressing "Yes" or "No", with "Yes" indicating a "1", or "true positive" and "No" a "0" or "true negative". The test items can be added if the user is not satisfied.

Testing. A user sets a kappa threshold and presses the "Run Test" button to run an IRR test. nCoder shows three test numbers: a kappa for the test set, a kappa for the training set (the test items in earlier cycles), and a rho value. The kappa values measure the agreement between the human coding and the machine coding. The rho shows whether or not the kappa value generalizes to the untested items. When rho <0.05, acceptable reliability is established between the researcher and the machine coding. In this study, we present results that address this cycle of the process. However, for a complete analysis, we recommend three pairwise IRR checks. First, IRR should be checked between rater one and the automated code to ensure proper classifier rules. Next, IRR should still be established between two human raters as is typical in common IRR processes to achieve good conceptual validity. As a final check, IRR testing should check reliability between the second rater and the classifier to make sure all human and computer raters agree on a concept.

Merging Training Data. If the test result is not satisfactory (i.e., the rho is above the alpha level, typically 0.05), the test items are moved to the training data set. The training data set contains all tested items. The user could review each item in the training set and the coding from the human and the machine.

Checking Disagreement. nCoder automatically checks the disagreement between the human coding and machine coding. The disagreed items are displayed to the user for investigation.

Refining Codes. The user may remove the disagreements by removing, adding, or refining regular expressions, resulting in changes of the regular expression list.

Updating Training Data. Each time the regular expression list is changed, the concept is re-coded by the machine and the machine codes in the training data set is updated. At the same time, the disagreements are changed. The refining-updating cycle may continue until the minimum number of disagreements is reached. Zero disagreement is possible but it may sometimes involve complicated regular expressions.

Preparing Output. The testing-refining cycle may be continued until the rho values for all concepts are less than 0.05. The output of nCoder will be the original data with added new concept columns containing machine coded values.

5 Kappa, Shaffer's Rho, Sample Size and Recall

To ensure reliability, nCoder relies on kappa values and Shaffer's rho. In this section, we will review concepts related to kappa and take a mathematical look at Shaffer's rho. At the end of this section, we will address an unsolved problem in nCoder: when the *base rate* (i.e., the ratio of the occurrences to the data size) of a code is low, the coding recall of the concept could be low or unacceptable, even if the kappa has been shown to be statistically significantly above a threshold with rho.

Kappa is a statistical measure proposed by Cohen in 1960 [15], which has been used widely as an inter-rater reliability measurement [6]. It measures the degree of agreement between two or more coders controlling for chance agreement. In the case of nCoder, we may consider items coded by a human rater as "1"s as "human positives" and "0"s as "human negatives". The machine coded "1"s and "0"s are "machine positives" and "machine negatives", respectively. For each concept, an item may have a pair of human-machine coding as "1-1", "1-0", "0-1", or "0-0". If we take the human coding as actual truth and computer coding as prediction, kappa becomes a measure of the performance of the machine coding. Following the notions in literatures, we denote the proportion "1-1" by *tp* (true positive), "1-0" by *fn* (false negative), "0-1" by *fp* (false positive), and "0-0" by *tn* (true negative). Kappa is often written as

$$\kappa = \frac{p_0 - p_c}{1 - p_c},$$

where $p_0 = tp + tn$ is the total agreement between the human and the machine, and $p_c = (tp + fp)(tp + fn) + (tn + fp)(tn + fn)$ is the chance agreement between the human and the machine.

nCoder aims at minimizing the number of items a human rater has to code in a test set for the purpose of establishing reliability and providing a warrant for validity. That is, nCoder was designed to help researchers use the smallest sample size possible when establishing inter-rater reliability. The question is, how do we know that the machine coding is good enough? In other words, how much data does a trained human rater need to code in order to establish that the machine could reliably code untested data with a high enough level of agreement with a trained human rater? Shaffer and his colleagues provided the rho measure to answer this question. Roughly speaking, a rho <0.05 in nCoder means that we would be wrong less than five percent of the time if we concluded that if both the human rater and the machine were to code all the data, not just the sample or test set, that agreement between their ratings would be greater than the threshold of interest. Details about the computations of rho could be found in Shaffer [1] and Eagan et al. [6].

When a code has a low base rate, it is often practical to check all machine coded occurrences and get a low false positive rate by refining the regular expressions. However, checking false negative is often impractical. For example, if the base rate is about 5% and the data size is 10,000. Then the user may go through 500 items with machine coded "1"s and see if any item is a false positive. However, to check the false negative, the user would need to check 9,500 items. Thus, nCoder users may often end up with a coding which is of very few, if any, false positives but potentially a high false negative rate. To illustrate this issue, let's consider the situation when the false positive rate is zero. In this case, the false negative rate is a function of true positives and the kappa:

$$fn = \frac{2tp(1 - tp)(1 - \kappa)}{2tp + (1 - 2tp)\kappa}.$$

Since when there are zero false positives, $tp = 0$ implies $\kappa = 0$, the above equation holds only for $tp > 0$.

Figure 2 shows the curves of the false negative rate as a function of true positives for kappa = 0.65, 0.80 and 0.90, respectively. As an example, let's look at case when kappa = 0.65 (the top curve in blue). The curve shows that the maximum false negative rate is about 18% when the true positive rate is around 40%. When the true positive rate is very low, the false negative is also low. For example, for kappa = 0.65, when true positive rate is 5%, the false negative rate is also about 5%. Both rates are low. However, the false negative rate is about the same as true positive rate. In other words, the number of occurrences reported by the machine could be the same as those missed by the machine.

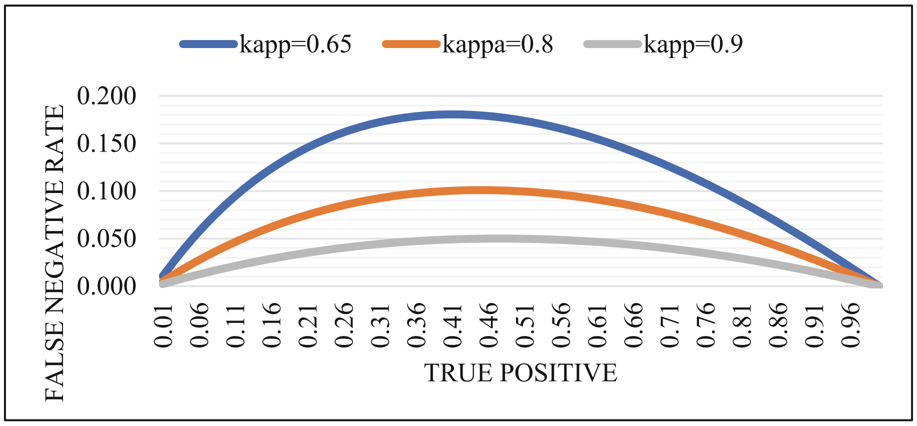


Fig. 2. False negative rate as a function of true positive and kappa when false positives = 0 (Color figure online)

Recall is a commonly used measure to indicate the capability of an algorithm or classifier in finding targeted items. Mathematically speaking, recall is the rate of true positives to the total truth, which can be written as:

$$recall = \frac{tp}{tp + fn}.$$

When the false positive rate is zero and $tp > 0$, for a given kappa, recall is a linear function of true positives:

$$recall = \frac{2(1 - \kappa)}{2 - k} tp + \frac{\kappa}{2 - \kappa}.$$

Figure 3 shows the recall lines as linear functions of true positives for kappa = 0.65, 0.80 and 0.90, respectively. When *tp* is small, the recall can be approximated by $\frac{\kappa}{2-\kappa}$. For example, when *tp* is small and kappa = 0.65, the recall is about 0.48. That means, when *tp* is small, a 0.65 kappa could not even guarantee a 50% recall.

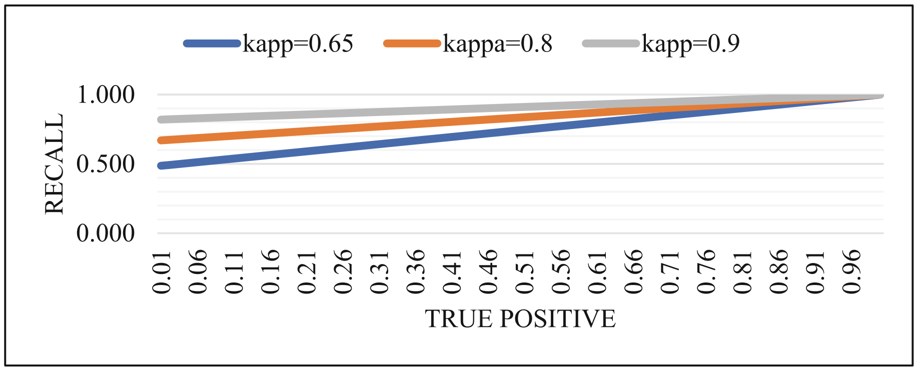


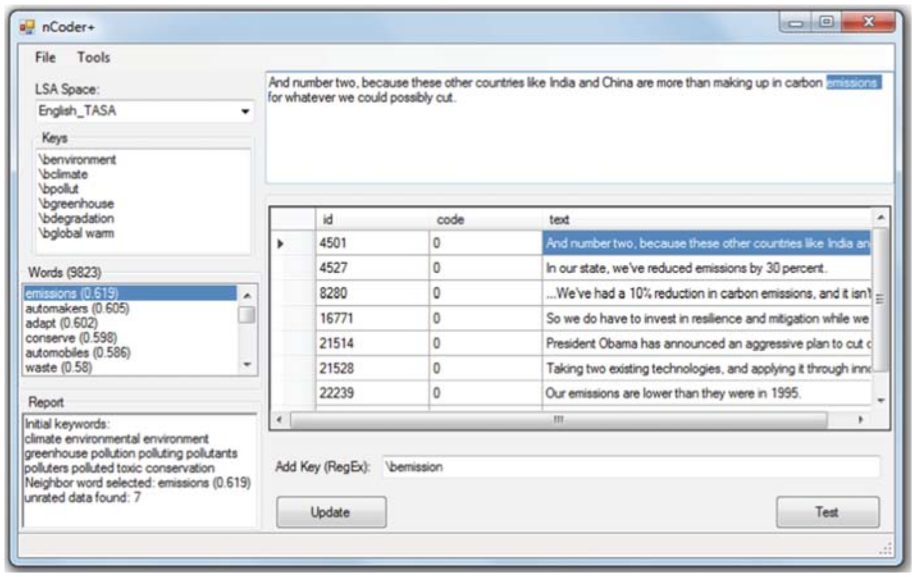
Fig. 3. Recall as linear a function of true positives for given kappa

6 nCoder+: Adding a Semantic Component to nCoder

To help improve recall when using nCoder, we built an add-on tool, called "nCoder+", that allows researchers to easily find false negatives, i.e., targets missed by nCoder. The tool is briefly described below and a real data example is used to demonstrate the tool's performance in the next section. In the final discussion section, we will talk about integrating this tool as a component into nCoder.

The core of nCoder+ is an LSA (Latent Semantic Analysis) component. LSA is a way to represent the "meaning" of words by vectors. Readers who are not familiar with LSA could refer to Landaure et al. [16]. The cosine between two vectors are often used to measure the similarity between two words. For a given word or phrase, the words with highest cosine values to the given word or phrase are called "nearest neighbors". Nearest neighbors make it possible to automatically find the most likely missing items.

Figure 4 shows a screenshot of nCoder+. On the left panel, there is a dropdown menu that allows user to select an LSA vector space. Below that is the "Keys" box that displays the regular expressions used to code the data. The "Words" box lists the neighbors to the keywords, together with the semantic cosine values as a measure of similarity of a neighbor to the keywords. The "neighbors" are sorted by similarity, so that the nearest neighbors are on the top of the list. The "Report" box below the neighbor list shows the result of each step. The table on the right panel shows the data under investigation. The box above the data table shows the content of any selected cell in the data table. The "Add Key" box is a place for users to enter new regular expressions. The "Test" button is used to check the validity of regular expressions. The "Update" button is used to add a new regular expression to the list.



LSA Space: English_TASA

Keys
\benvironment
\bclimate
\bpollut
\bgreenhouse
\bdegradation
\bglobal warm

Words (9823)
emissions (0.619)
automakers (0.605)
adapt (0.602)
conserve (0.598)
automobiles (0.586)
waste (0.58)

Report
Initial keywords:
climate environmental environment
greenhouse pollution polluting pollutants
polluters polluted toxic conservation
Neighbor word selected: emissions (0.619)
unrated data found: 7

And number two, because these other countries like India and China are more than making up in carbon emissions for whatever we could possibly cut.

id	code	text
4501	0	And number two, because these other countries like India an
4527	0	In our state, we've reduced emissions by 30 percent.
8280	0	...We've had a 10% reduction in carbon emissions, and it isn't
16771	0	So we do have to invest in resilience and mitigation while we
21514	0	President Obama has announced an aggressive plan to cut c
21528	0	Taking two existing technologies, and applying it through inn
22239	0	Our emissions are lower than they were in 1995.

Add Key (RegEx): \bemission

Fig. 4. Screenshot of nCoder+

nCoder+ starts from loading the regular expressions used in nCoder coding of a key concept and a csv data file with 3 columns: an id column, a text column and a binary code column. A "1" in the code column indicates that the text on the row contains the specific key concept.

After the regular expressions and the csv data file are loaded, nCoder+ extracts words from the text column and form a vocabulary. The words that match any one of the regular expressions are then identified as "keywords". For each vocabulary word, the LSA cosine values between this word and each of the keywords are computed. The maximum of these cosine values is taken as the similarity of the word to the keywords. Of course, if the word itself is a keyword, the similarity will be 1. The vocabulary words are then sorted by the similarity so that the nearest neighbors are on the top of the sorted word list. The word list is then displayed in the "Words" box with keywords excluded.

A user reviews the nearest neighbors from the top to see if there is any word that is highly associated with the key concept. If a word is identified, the user may click on it. All text items in the data with code "0" that contain the selected word will appear in the data table. The user may check the text items in the table and see if there are any missing items that should be coded.

When missing items are identified, the user may compose a new regular expression based on the selected word. To check whether the regular expression is well composed, the user may press the "Test" button. Then text items that match the new regular expression will be displayed in the data table. The user may revise the regular expression until it accurately flags the missing items.

Once the regular expression passes the test, the user may click on the "Update" button to add the regular expression to the regular expression list. The matched text items will be coded as 1. The selected word will be added into keywords and removed from the displayed word list. The similarity of the words will be re-computed, re-sorted and displayed as neighbors to the new keywords. This procedure is repeated until the user cannot find any new words in the word list that could be used to find possible missing items.

7 nCoder+ Validation

The "Primary Debates 2016" dataset was used to test the tool. The dataset contains transcripts of every debates held during the 2016 primary season. The scripts were split into 23,714 utterances. One of the co-authors ran nCoder to code the concept "Environmental Issues", ended up with the regular expressions shown in Table 1.

Using base rate inflation of 0.20, the concept of "Environmental Issues" was validated with a kappa of 0.8 and a Shaffer's rho 0.01. That guarantees that the coded values could have at least 0.65 kappa with a human rater if the human rater had rated the whole data set. The final data set flagged 120 utterances that contain the key concept. Another co-author checked all 120 utterances and found that the false positive rate was 0. However, the recall was not checked because that would mean to go through the rest of the 23,596 utterances and see if there are any more utterances that contain the key concept. With zero false positive, 0.65 kappa, and true positive = 120/23, 596 = 0.5%, the corresponding recall is about 0.5, which means that there could be as many as 120 items containing the "Environment Issues" that were not coded as "1".

The data and the regular expressions were loaded to nCoder+. A set of keywords were extracted from the regular expressions and the rest of the words were displayed in the order of semantic similarity to the keywords. The first new keyword we identified was "emissions". The tool showed up 7 items containing "emissions" that should be but were not coded as "1". The regular expression "\bemissions" was added to the regular expression list and 7 new items were added to true positives. Repeating this process, we identified 8 new keywords as shown in Table 2. nCoder+ showed 175

Table 2. Keywords, added regular expressions and additionally identified items

Neighbor word	Items involved	RegEx	Items coded
Emissions	7	\bemissions	7
Clean	39	\bclean.*(water|power|electric|energy)	23
Gases	1	\bgases	1
Gas	34	(\bgas.*energy)|(\benergy.*gas)	4
Waste	21	\bwaste.*water	2
Coal	60	(\buse of coal)|(\bcoal\b.*(energy|clean))	2
Carbon	4	\bcarbon	4
Solar	9	\bsolar	9
Total	**175**		**52**

items containing these 8 new keywords. With 8 carefully generated regular expressions, 52 new items were added to the true positives. Thus, assuming there were a total of 240 positive items, the recall is increased from 50% to 172/240 = 72%.

8 Discussions

nCoder is a tool that helps both qualitative and quantitative researchers in coding text data. The regular expression based automatic coding is simple and effective. When the data size is large, it is usually hard to define a complete set of regular expressions that represent a certain concept, which can result in low coding recall. This paper provided an effective tool, nCoder+, to find missing items and thus improve recall. The current nCoder+ tool is just a prototype for validating the idea. There are multiple ways to integrate this tool into nCoder. One way is to add it as a post process component and use it in the way we used in this paper. The other way is to integrate it as a component in the nCoder's testing process (see Fig. 1). At the nCoder testing step, a test set is sampled from data that has not been exposed to the researcher. A proportion (e.g., 20%) of the test items are sampled from the data with machine coded "1"s and the rest are randomly sampled from the remaining data. To make use of the semantic neighbors, a proportion of the "0" items could be those that contains nearest neighbors of the keywords. Thus, the test set contains three types of items: "1" items, "near neighbor" items and "remote neighbor" items. The "1" items will help determine the true positive; the "near neighbors" will help improve recall; and the "remote neighbors" will help determine the true negative.

This paper only considered integrating the idea of semantic neighbors into nCoder. Other NLP methods may also be useful to further improve nCoder. For example, topic modeling could help researchers to find meaningful concepts that researchers may otherwise miss. Neural network algorithms may also help in inferring concepts from given keywords. Our future work will be continuously incorporating latest advances in natural language processing and providing researchers with better coding tools.

Acknowledgements. The research was supported by the National Science Foundation (SBR 9720314, REC 0106965, REC 0126265, ITR 0325428, REESE 0633918, ALT-0834847, DRK-12-0918409, 1108845; DRL-1661036, 1713110; ACI-1443068), the Institute of Education Sciences (R305H050169, R305B070349, R305A080589, R305A080594, R305G020018, R305C120001), the Army Research Lab (W911INF-12-2-0030), and the Office of Naval Research (N00014-00-1-0600, N00014-12-C-0643; N00014-16-C-3027), the Wisconsin Alumni Research Foundation, and the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin-Madison. The opinions, findings, and conclusions do not reflect the views of the funding agencies, cooperating institutions, or other individuals.

References

1. Shaffer, D.W.: Quantitative Ethnography. Cathcart Press, Madison (2017)
2. Chi, M.T.H.: Quantifying qualitative analyses of verbal data: a practical guide. J. Learn. Sci. **6**, 271–315 (1997)

3. Saldaña, J.: The Coding Manual for Qualitative Researchers (2014). https://doi.org/10.1007/s13398-014-0173-7.2
4. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Transaction, New Brunswick (1967)
5. Charmaz, K.: Constructing Grounded Theory. SAGE, London (2006)
6. Eagan, B.R., Rogers, B., Serlin, R., Ruis, A.R., Irgens, G.A., Shaffer, D.W.: Can we rely on IRR? testing the assumptions of inter-rater reliability. In: CSCL 2017 Proceedings, pp. 529–532 (2017)
7. Blei, D.M., Edu, B.B., Ng, A.Y., Edu, A.S., Jordan, M.I., Edu, J.B.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003). https://doi.org/10.1162/jmlr.2003.3.4-5.993
8. Hu, Y., Boyd-Graber, J., Satinoff, B.: Interactive topic modeling. In: Proceedings of the 49th Annual Meeting Association for Computational Linguistics Human Language Technologies, pp. 248–257 (2011)
9. Marquart, C.L., Swiecki, Z., Eagan, B., Shaffer, D.W.: ncodeR (Version 0.1.2) (2018)
10. Eagan, B.R., Rogers, B., Pozen, R., Marquart, C., Shaffer, D.W.: rhoR: Rho for inter rater reliability (Version 1.1.0) (2016). https://cran.r-project.org/web/packages/rhoR/index.html
11. Gašević, D., Joksimović, S., Eagan, B., Shaffer, D.W.: SENS: network analytics to combine social and cognitive perspectives of collaborative learning. Comput. Hum. Behav. **92**, 562–577 (2019)
12. Cai, Z., Pennebaker, J.W., Eagan, B., Shaffer, D.W., Dowell, N.M., Graesser, A.C.: Epistemic network analysis and topic modeling for chat data from collaborative learning environment. In: Proceedings of the 10th International Conference on Educational Data Mining, pp. 104–111 (2017)
13. Sullivan, S., et al.: Using epistemic network analysis to identify targets for educational interventions in trauma team communication. Surg. (United States) **163**, 938–943 (2018). https://doi.org/10.1016/j.surg.2017.11.009
14. Shaffer, D.W., Ruis, A.R.: Epistemic network analysis: a worked example of theory-based learning analytics. In: Handbook of Learning Analytics Data Mining, in press (2017)
15. Cohen, J., Cohen, J.: A coefficient of agreement for nomial scales. Educ. Psychol. Meas. **20** (1), 37–46 (1960). https://doi.org/10.1177/001316446002000104a coefficient of agreement for nomial scales. Educ. Psychol. Meas. 20, 37–46 (1960). https://doi.org/10.1177/001316446002000104
16. Landauer, T., McNamara, D., Dennis, S., Kintsch, W.: Handbook of Latent Semantic Analysis (2007)