Collaborating with the Machines
A hybrid method for classifying policy documents

Matt W. Loftis
Aarhus University
mattwloftis@ps.au.dk*

Peter B. Mortensen
Aarhus University
peter@ps.au.dk

January 2018

Abstract

Governments produce vast and growing quantities of freely available text: laws, rules, budgets, press releases, etc. This information flood is facilitating important, growing research programs in policy and public administration. However, tightening research budgets and the information's vast scale force political science and public policy to aspire to do more with less. Meeting this challenge means applied researchers must innovate. This paper makes two contributions for practical text coding – the process of sorting government text into researcher-defined coding schemes. First, we propose a method of combining human coding with automated computer classification for large data sets. Second, we present a well-known algorithm for automated text classification, the Naïve Bayes classifier, and provide software for working with it. We argue and provide evidence this method can help applied researchers using human coders to get more from their research budgets, and we demonstrate the method using classical examples from the study of policy agendas.

Published: Policy Studies Journal

*The authors wish to thank Jason Eichorst, Henrik Bech Seeberg, Martin Bækgaard, Carsten Jensen, and Public Administration workshop participants at the Aarhus University Department of Political Science.

Often scholars of public administration or public policy work on research problems that require them to classify or categorize huge numbers of documents. Furthermore, classification often needs to be consistent across a large number of units of analysis and/or across time. One example is the literature on rule dynamics, where scholars need to categorize whether a given set of rules belongs to one category or another based on short text summaries. For instance, does this law or that specific paragraph belong to the domain of energy policy, education policy, social policy, or environmental policy (see Jakobsen and Mortensen, 2015; March, Schulz and Zhou, 2000; Schulz, 1998; van Witteloostuijn and de Jong, 2007; Zhou, 1993).

Another example is the structure of government strand of research, which focuses on the design, re-design, termination and birth of public agencies and/or ministries (Carpenter, 2001; Lewis, 2003; Yesilkagit and Christensen, 2010). In this case, the challenge of categorization is to consistently classify organizations over decades and/or cross-nationally into different policy areas or jurisdictions. A third example is public budgeting research. As it is well-known, budget formats change from time to time making it challenging to systematically trace how spending on a given government function has evolved over time (Jones, Zalányi and Érdi, 2014). To address such challenges, researchers need to carry out tedious, manual categorizations of clusters of spending based on text descriptions of budget items (see True, 2009). Other examples of research relying on large scale classifications of government text can be found within the Institutional Analysis and Development framework (see Ostrom and Schlager, 2014), within media and communication research (Thesen, 2013; Vliegenthart et al., 2016), as well as within a broader range of research relying on classification of political activities such as congressional hearings, bills, parliamentary questions, party manifestos, etc. (Adler and Wilkerson, 2013; Green-Pedersen and Walgrave, 2014).

The goal of this paper is to propose a simple and practical approach to computer-assisted data coding for public administration and public policy scholars classifying large amounts of text and to provide a practical guide for executing such analyses. We argue and provide evidence that this practice can help overcome significant challenges facing

researchers' capacity to code large amounts of data and maintain the data at the center of their research program. Budget constraints limit opportunities to launch large-scale human data collection, and a combination of limited experience with automated coding and time constraints prevent many applied policy scholars from applying even well-known automated coding tools on a broad scale.

Thus, the intended audience of this paper is not the small group of scholars with the training and competence in using advanced tools for automated coding, but the large group of public administration and public policy scholars whose research could benefit significantly from computer assisted text classification. These are the researchers who would normally rely entirely on human coding either based on their own preferences or due to various time and resource barriers to adopting tools for automated coding.

The alternative we introduce and advocate in this paper is a human-computer collaboration approach designed to ease budget constraints by speeding the coding process and dramatically reducing the amount of human coder time required to build large datasets of coded data. Furthermore, including computers in the work flow means humans do far less actual coding and move primarily into the role of applying expert knowledge to improve computer coding reinforcing the role of expert knowledge in the coding process.

We illustrate the steps in and the virtues of the approach using data from the Comparative Agendas Project (CAP). As illustrated by almost 60 articles published in Policy Studies Journal and a total of nearly 400 publications according to the latest count (Baumgartner, Jones and Mortensen, 2017, 85), this is a vivid and growing research field. The CAP project is a network of researchers united around a demanding measurement system developed to classify a broad range of political activities into topics which can be compared over time and across political systems (see Jones, 2016). Thereby, it is also a research field where the challenges sketched above are very real.

First, content coding policy agendas is detailed: the CAP coding scheme consists of 21 major topics and more than 200 subtopics. Second, the text material used in agenda-setting research is quite heterogeneous, varying from legislative hearings to speeches by heads of state, legislative debates, parliamentary questions, and more. The scale and

diversity of this data makes coding new policy agendas data both labor intensive and time consuming. Third, the material to be coded is enormous. The U.S. Congressional Bills Project consists of more than 450,000 titles of proposed legislation (Adler and Wilkerson, 2015), while the larger U.S. Policy Agendas Project maintains data on thousands of Congressional hearings, entries in the Congressional Quarterly Almanac, executive orders, State of the Union addresses, Supreme Court rulings, budgetary allocations, and more (Baumgartner and Jones, 2016).

The Danish Policy Agendas Project, for instance, includes almost 13,000 legislative bills since 1953, along with thousands of interpellations, motions, parliamentary questions, prime ministerial speeches, and others (Green-Pedersen and Mortensen, 2013). The broader Comparative Agendas Project includes data on thousands of bills, speeches, and other expressions of the policy agenda in more than a dozen countries and several U.S. states, which have together taken years of labor to curate and classify. Lastly, the data addressed by the CAP expands daily as governments pass laws, hold hearings, approve budgets, etc.

The costly and time-consuming process of collecting and content coding policy agendas data presents a major barrier to initiating new country data collection projects, and the scale of the data, already vast, grows more quickly over time as governments make more and more information available online. At the same time, social science research funding is tightening around the world (Plazek and Steinberg, 2013), making it increasingly difficult to mobilize the funding needed to engage expert human coders to prepare large data sets. Currently, 15 CAP country projects exist, relatively few in comparison to other cross-national research programs.[1] A still further difficulty for the CAP is the need to update existing data. Keeping a corpus of policy agendas data up to date as new data is constantly added can require new funding, and grants for updating existing datasets can be more difficult to secure than those for preparing new data. Substantially, this threatens to limit the relevance of agenda-setting research. Today, several major agendas

[1] See, for example, the 60 countries included in the Comparative Manifesto Project corpus (https://manifesto-project.wzb.eu/).

datasets end five or ten years ago. Given that the dynamics and drivers of policy agenda setting likely change over time (Mettler, 2016), this is a lurking barrier to further scientific investigation of government attention.

In this paper, we suggest practices for confronting these challenges efficiently and affordably, without compromising data quality. Specifically, we advocate and outline a procedure for human-computer collaboration for coding policy and public administration data. Along the way, we introduce to the reader and apply a machine learning algorithm not widely used in the subfield and demonstrate its performance on real data.[2] Finally, we demonstrate in concrete terms how these tools can maximize returns on research budgets.

Practical practices for text coding

As argued above, the basic challenge facing policy agendas scholars is of relevance to a broad range of research in public administration and public policy. We use examples drawn from the study of policy agendas to illustrate a method and workflow for computerized text coding,[3] not least because the challenges are very salient to this field of research and the potentials for improvement equally high.

Historically, CAP projects have employed mostly or only human coders to read agendas text documents and classify them into major and minor topics. We surveyed CAP country project leaders in June 2016 about their application of automated tools to date. Responses indicate many projects have experimented with computerized classification, though only one reported using automated tools as a key part of their work. We believe the written comments from the survey respondents reflect the experiences of many public policy and public administration scholars:

[2] Software used in all analyses is freely available online at `https://cran.r-project.org/` and `http://github.com/mattwloftis/agendacodeR`. Replication code available on the author's website.

[3] We use the terms "text coding" and "text classification" interchangeably. Coding is generally used for human labor (i.e. codebook and human coder), while classification is favored in the computer science literature on categorizing text.

- *"...I think we need more training both on the statistical concepts and the software technicalities to get to use these tools in a more systematic way"*
- *"The predicted codes were not satisfactory, once checked by our coders. In the end, we gave up."*
- *"We compared computer coding results against previous-double hand coded and reconciled observations. We found the computer coded data to not reach our standards of accuracy for it to be worth while to implement on a broad level."*
- *"None [i.e. no computerized classification methods] thus far made it to more permanent use, mainly financial resource limitations for really developing the tools further."*

Among CAP projects using computerized methods, the modal tool applied was the `RTextTools` package (Jurka et al., 2013). As Jurka et al. (2013) note, `RTextTools` is recommended for data sets no larger than around 30,000 documents. The memory demands of the methods it utilizes[4] can lead to long processing times even on newer desktop computers. With larger data sets, for example those including a hundred thousand or more observations, this limitation can lead to prohibitively long processing time and crashes.

This leads to the core of our argument about the challenges facing data collection and data maintenance efforts in the the wider fields of public administration and public policy studies. Budget constraints limit opportunities to launch large-scale human data collection, and computing power and time constraints limit the scope for applying sophisticated automated coding tools on a broad scale – both because these methods are demanding on computing resources and they are complex enough to require expert knowledge.

The answer we propose to these challenges is two-fold. First, we advocate human-computer collaboration in the coding. This eases budget constraints by speeding the coding process and dramatically reducing the amount of human coder time required to build large policy data sets. Furthermore, including computers in the work flow means humans do less actual coding and move primarily into the role of applying expert knowledge to improve computer coding – reinforcing the role of expert knowledge in the coding

[4] These include many advanced methods we do not discuss: e.g. support vector machines, multinomial logistic regression (maximum entropy), or neural networks (Jurka et al., 2013).

process. In the next section, we present this work flow in detail.

The second part of our proposal for overcoming the difficulties facing policy data coding is to recommend one particularly transparent, fast, and mathematically simple approach to computer coding, namely a multiclass Naïve Bayes algorithm with Bernoulli features. We describe in general terms how to apply it in the next section, and our motives for choosing Naïve Bayes and a detailed explanation of the method are discussed later. We also provide and describe free purpose-built software for implementing it which includes tools to help analysts scrutinize computer results so they can refine the codings the computer provides and improve their accuracy.

After introducing the work flow and our preferred method, we illustrate our human-computer hybrid method of coding policy agendas data with a brief description of the process of coding a relatively large policy agendas data set. The data consist of the text of all items appearing on Danish municipalities' city council meeting agendas over a period of several years. Furthermore, we check the performance of our recommended computer coding method against a well-known benchmark data set, the titles of U.S. bills, maintained by the U.S. Congressional Bills Project (Adler and Wilkerson, 2015).

Collaborating with a machine

We describe here a general work flow for human-computer collaboration in text classification aimed at newcomers to computerized text analysis. We will use examples of coding policy agendas data into subtopics, though the procedure applies to any task involving categorizing public administration or public policy text. Researchers with the resources and expertize to apply more sophisticated classification methods may find those achieve greater computer accuracy than our method. However, even in those cases, we consider our proposal a useful starting point before moving on to apply more advanced tools.[5] The

[5] For interested readers, we briefly discuss more sophisticated computer classification methods in appendix, including the state of the art in human-computer collaborative approaches.

process we propose consists of three stages: the human-coding stage, the collaborative stage, and finally the pure machine-coding stage.

Stage one: To begin any supervised machine classification, a set of human-coded documents is necessary. This is called the "training set." The human-coding stage refers to preparing the training set. For our examples, taken from CAP, this process is the standard coding procedure laid out in the master codebook and accompanying materials.[6] For our method, the distribution of codings, or classes, in the training set should resemble the distribution in the uncoded data, so it is best to start with a random sample. As a rule, larger training sets are better, especially in the context of CAP coding since there are around 200 subtopics – some occurring very rarely. As we show later, at least 500 items is a reasonable starting point, though several thousand training items provide a much stronger start. With an initial set of correctly coded documents in hand, the computer can get involved.

Stage two: The second, collaborative, stage is iterative. This stage is repeated until the desired level of accuracy is achieved. In this stage, the computer "learns" from the training data and then classifies all uncoded documents. Once uncoded data has been assigned a (preliminary) class, the analyst takes a random sample from this newly machine-coded data and human coders review it for accuracy. Early on this accuracy can be quite poor. Human coders make corrections where needed. The random sample for human review can be of any size – though larger samples yield larger increases in computer performance.

This human coder review accomplishes three tasks. First, it checks the computer's performance on virgin data – i.e. data the computer has not seen – to measure out-of-sample predictive success. Second, information gleaned from examining the computer output can be used to refine the information the computer uses to classify documents, boosting the accuracy of subsequent machine coding. We discuss procedures for this later.

[6] Find the CAP master codebook at: `http://www.comparativeagendas.net/pages/master-codebook`

Third, the human-reviewed data is added to the training set, growing the information used to train the computer.

Following human review, the analyst trains the computer again using the enlarged training set, boosting the accuracy of the next machine coding. New computer-assigned codes are generated and the collaborative step is repeated.

Stage three: When the accuracy of the computer-assigned codes converges to a level within the acceptable bounds for normal reliability checks, the collaborative stage ends and computer-only coding begins. In this final stage, the computer classifies all remaining virgin data and coding is considered complete. This step can be augmented with a final human review. To make the final human review more efficient, the analyst can concentrate on reviewing only classes in which the computer performs less well. We provide assessment tools to identify problematic classes.

To sum up, consider how working with the computer speeds and strengthens the coding process. First, in our experience, computer coding achieves accuracy rates approaching 50% within a few iterations of stage two. By accuracy, we refer to the proportion of virgin documents the computer assigns to the correct class. Some classes will be easier to identify than others, so computer classification often identifies certain classes with near perfect accuracy. Second, we find human coders are able to review computer-coded data more quickly than they can code virgin data, and they perform even better as they review more accurate computer classifications. This yields three benefits: a) reassigning human coders to review computer codes directly speeds coding; b) human reviewers work increasingly quickly as coding proceeds; and c) human review of "easy" classes can move extremely quickly.

Finally, computer classification solves the problem of updating data sets. The trained computer classifier can be stored for later and used to classify new data as it appears with minimal additional time and effort.

Classifying with limited computing power

The choice of classification method, as in standard statistical modeling, is first motivated by inferential goals. In our case, we face a familiar problem with many possible solutions. The aim is to classify documents[7] into classes,[8] based only on the documents' text. Therefore, we first assume each document's class is some function of its words. A wide variety of models and machine learning algorithms can model document classes as a function of words (Grimmer and Stewart, 2013; Jurka et al., 2013; Munzert et al., 2015). Three characteristics of much agendas and policy data influence our choice: (1) the number of possible classes; (2) the number of different words that are relevant; and (3) the number of documents in the data. All three are quite large for the standard CAP country project and for many policy studies applications.

Classifying text implies treating words or phrases as numerical data (Grimmer and Stewart, 2013). That is, we begin by generating a matrix in which documents are observations (rows). The variables or columns (called *features* in the language of classification methods) are indicators, counts, or some other measure of the appearance or salience of words or phrases in documents. The data have as many columns as there are words or phrases in the entire corpus of documents. A data set of tens of thousands of documents can easily generate as many, or more, features as observations. This can leave a statistical model unidentified, or at least constitute a hefty computational burden and therefore create long processing times for many models.

Furthermore, many policy studies applications code documents into many classes. For example, the CAP codebook maps agendas into around 200 subtopics. Discrete choice models can be estimated with so many classes (Train, 2009), but estimating such a model using data the size of standard CAP data sets can be computationally unmanageable. Classifying multiclass data is computationally easier with algorithmic approaches, but many of these perform poorly with highly unbalanced classes (Japkowicz and Stephen,

[7] We use the terms document or observation interchangeably here to refer to a single text, legislative bill title, budget category, etc. in the data.

[8] i.e. subtopic, major topic, or other category

2002) — that is, when certain classes appear very frequently and others very rarely. This is an endemic problem for much policy data. Political attention is never distributed evenly across all policy areas, and certain policy areas appear very often while others come up rarely. Therefore, our method must perform well despite classifying unbalanced data.

A final factor in choosing a classification method are constraints imposed by researchers' time and budget. To be useful to applied scholars, a classification method should produce results quickly on a desktop computer without requiring the use of, say, cloud computing or other advanced computing tools to achieve results quickly. This narrows the field of choices substantially. A variety of sophisticated approaches are applicable to classify policy text, but sophistication comes at costs in time and computing resources. For example, support vector machines, neural networks, and other tools are popular and proven text classification tools, but can make massive memory demands on the computer even for small jobs.

With this in mind, we advocate Naïve Bayes as a computationally frugal text classification algorithm. The Naïve Bayes classifier (NB) estimates the probability that a document, d, comes from class c as a function of its words and/or phrases. Probabilities are estimated via Bayes' rule. Training data (i.e. coded documents) establish prior probabilities of observing certain classes ($P(c)$), certain documents ($P(d)$), and certain documents in each class ($P(d|c)$). Combining this prior information allows us to estimate the probability that new documents fall into each class:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

NB is not a single method. Rather, it comes in various forms for different purposes and data types (Metsis, Androutsopoulos and Paliouras, 2006). Their commonality is that document features are assumed to vary independently of one another, net of the class in which they appear. This assumption is the naïve part. It is likely, for example, that documents mentioning nurses also mention hospitals. Nevertheless, ignoring associations between features makes the algorithm fast and efficient.

Varieties of NB are widely applied to classify text (Lewis, 1998; McCallum and Nigam, 1998; Hovold, 2005; Kibriya et al., 2005; Schneider, 2005; Chen et al., 2009). These implementations vary in the form of priors, distributional assumptions, measurement of features, and means of selecting which features of the training data to use – resulting in some variety in their computational intensity. We adapt a particularly efficient version of NB for multiclass data – that is, data falling into more than two classes (O'Neil and Schutt, 2013).

Details of our own estimation are found in appendix A. In short, we represent each document as a vector of 1s and 0s for each feature appearing in any document – a 1 means a feature appears in the document one or more times, and a 0 means the feature does not appear in the respective document. Then, we calculate the prior probabilities of observing each class ($\hat{\theta}_c$) and of observing each feature in each class ($\hat{\theta}_{jc}$). With these priors, we can calculate the posterior probability that a new document falls into class c, given its features. We calculate this for each document and every possible class.

Several points merit discussion. As in other NB applications (see Schneider, 2005), we find performance is best when considering only the appearance or absence of features, discarding information about additional appearances of the same word in a single document. Note also the estimation of $\hat{\theta}_{jc}$ and $\hat{\theta}_c$. Prior *class* probabilities, $\hat{\theta}_c$, are simply the number of training documents in class c divided by the number of training documents. However, we smooth values of prior word-in-class probabilities, $\hat{\theta}_{jc}$, to eliminate zero and one prior probabilities (Frank and Bouckaert, 2006). Smoothing $\hat{\theta}_{jc}$ is necessary because some features appear extremely rarely in certain classes. Consider an example. If training data contain no documents about foreign policy mentioning doctors then the prior probability that new documents containing the word doctors pertain to foreign policy would be zero. This means new documents about foreign aid for medical assistance that mention doctors would have zero probability of relating to foreign policy. This is obviously mistaken and smoothing prevents these mistakes.

This approach to NB text classification has several advantages for researchers classifying relatively large and highly unbalanced multiclass data – exactly the type commonly

studied in policy and public administration. Foremost are speed and accuracy. NB estimates class and feature priors – $\hat{\theta}_c$ and $\hat{\theta}_{jc}$, respectively – by simply tabulating over the training data and smoothing. This process is very fast relative to fitting a statistical model or using complex algorithms. Once NB is trained, predicting classes for a large set of new documents requires a quick matrix multiplication and some addition. This process is quite fast. At the same time, NB's wide application in text classification problems attests to its accuracy.

A final benefit is that the algorithm is completely transparent. This transparency is an important virtue compared to other and more sophisticated machine learning methods.[9] Analysts can easily track associations between features and classes and see how NB arrives at its answers. This is helpful for troubleshooting, justifying the face validity of computer codes, and tuning the algorithm to improve performance. In the next sections, we will present empirical examples from policy agendas data to demonstrate its performance. Before moving on to applications, however, we briefly discuss how analysts can optimize the performance of this basic algorithm.

Tuning for performance

Text classification for policy data aims to accurately forecast documents' classes. As such, the only principled guide to making decisions about the analysis is to determine what yields better accuracy. In our case, at least two such choices merit discussion: (1) regularization of posterior probabilities, and (2) selection of features. We consider them in turn and discuss how to optimize predictive performance.

As described above, NB classifies documents into classes by calculating posterior probabilities that a given document belongs to each class. That is, for each classified document, it returns a vector of probabilities as long as the number of categories. One standard way to classify documents is to select the class with the highest posterior probability for each document (Lewis, 1998). However, any number of alternatives might be

[9] One of the most promising alternatives is likely "active learning" combined with support vector machines (SVM) and we discuss this alternative in appendix.

useful. One method we implement is to take the ratio of the posterior probabilities to the simple class priors for each classified document and then to classify documents into the class with the highest ratio – i.e. the class whose probability increased the most relative to the class prior after observing the document. This regularization helps ameliorate the difficulty of classifying highly unbalanced multiclass data by removing some bias toward high frequency classes.

Additional gains in predictive accuracy can be achieved by refining the terms included in the analysis – that is, by deleting or merging certain features. NB generally performs better when using fewer, more informative, features. We refine training features in three ways. First, we apply standard refinements such as removing so-called stop words and stemming. Stop words refer to common words unlikely to vary by class, such as articles, prepositions, pronouns, etc. Stemming refers to transforming words into their root, so that, for example, "pass," "passed," and "passing" are treated as instances of the same word: "pass."

Second, in line with our human computer collaboration approach, we recommend applying expert human knowledge to refine words in the training vocabulary. This can vary in sophistication. On the unsophisticated side, it is often useful to delete features like months of the year, days of the week, many geographic locations, numbers, punctuation, etc. Something more advanced might involve merging acronyms and abbreviations with their spelled out forms. Refinements requiring expert knowledge might involve merging words used interchangeably in policy discussions, merging features naming the same agency or program, or standardizing policy language that has evolved over time. Such refinements must be made with care because deleting features that convey information can as easily damage as improve accuracy. Likewise, merging terms that convey inconsistent information can harm accuracy. With any refinement, it is recommendable to test for what increases performance.[10] NB's transparency is helpful in applying expert judgment to feature refinements. As we will demonstrate, analysts can easily view which classes

[10] For a more detailed discussion of refining language for text analysis, see Lucas et al. (2015).

features relate to and trace how features contributed to the codings assigned by the computer.

Third, and finally, it is useful to delete the least informative features from the training data. We do so by calculating the "mutual information" between features and classes – i.e. the degree to which the variation in feature appearance discriminates classes from one another – using the measure described in McCallum and Nigam (1998).[11] In the examples that follow, we drop around the least informative 75% of features remaining in the training set after the above refinements.

Making the best tuning choices requires trial and error and in appendix C we briefly discuss the two standard approaches to validate one's choices.

Illustrative case: Danish city council agendas

We illustrate how this process works in practice with an example of a policy studies data collection project. The Causes and Policy Consequences of Agenda Setting (CAPCAS) project investigates why societal problems gain or lose attention on political agendas and how this agenda-setting process matters for policy decisions. Project data consist of the text of all agenda items discussed in all Danish municipalities' city council meetings over a period of around seven years. Following past agenda setting studies, CAPCAS converted all agenda text to digital format and trained human coders to apply numeric codes to each item corresponding to a version of the comparative agenda setting project codebook modified for local government. The data include over 200,000 agenda items, each about one sentence long. Given the size of the task, two years of the four-year project were originally budgeted to data preparation.

In the first year, the project assembled the complete uncoded data and hand coded 24,000 randomly selected observations. At this time, CAPCAS began considering computer classification. Budget and time constraints were primary motivations. Although there were sufficient resources to use human coders as planned, the process was slow and the project aimed to increase its scope. By adding the agendas of city council committees,

[11] See appendix C for full details.

of which there are several per city, CAPCAS could achieve a more nuanced understanding of the agenda setting process in the councils. However, this required coding an unreasonable additional number of agenda items: around half a million.

After an initial investment in choosing and refining a computerized method, the project aimed to classify the additional data at virtually zero marginal cost. Since a detailed coding scheme and initial training data already existed and classification methods had been applied in political science already (Yu, Kaufmann and Diermeier, 2008; Hillard, Purpura and Wilkerson, 2008; Workman, 2015; Burscher, Vliegenthart and De Vreese, 2015), this became a major focus.

As reviewed above, computational simplicity and transparency led to the selection of NB for classification. The final data set was too large by the standards of desktop computing to apply more complex tools. At this scale, some tools proved unusable due to long processing times, and even with quite small subsets of the data certain methods struggled to handle the highly unbalanced multiclass agendas data. Even with CAPCAS' large initial training data, many subtopics and many words appeared very infrequently. Thus, we developed the system of moving from computer classification to human review to expert refinements to improve the classifier.[12]

Using the workflow and method described above, the project completed council agenda coding six months early and under budget, and an additional approximately 500,000 city council committee agenda items were added to the data. We demonstrate here the classifier's accuracy on around 200,000 agenda items from approximately 9,600 Danish city council meetings in all 98 municipalities. We divide the corpus in three parts: training with around 120,000 items and dividing remaining data into test and holdout data. Dan-

[12] Human refinements were especially important in this non-English language text. Standard word stemming tools for Danish text made many errors, stemming many specialized terms incorrectly. Important mistakes had to be corrected and features with identical meaning but dissimilar spelling grouped into single features. This eventually resulted in around a 20% reduction in the number of features (from around 55,000 to around 44,000 for the main council agendas data).

ish language word stemming and human feature refinements developed in the CAPCAS project are applied. We further refine features by eliminating those below about the 84th percentile of mutual information with the training set classes. This threshold was chosen by optimizing predictive accuracy on the test data. To demonstrate the usefulness of this step, figure 1 plots accuracy rates over different thresholds of mutual information for removing features. For illustrative purposes, we also plot accuracy both using our suggested regularization of posterior probabilities (ratio of posterior to prior class probabilities) and using the standard approach of assigning documents to classes with maximum posterior probability.

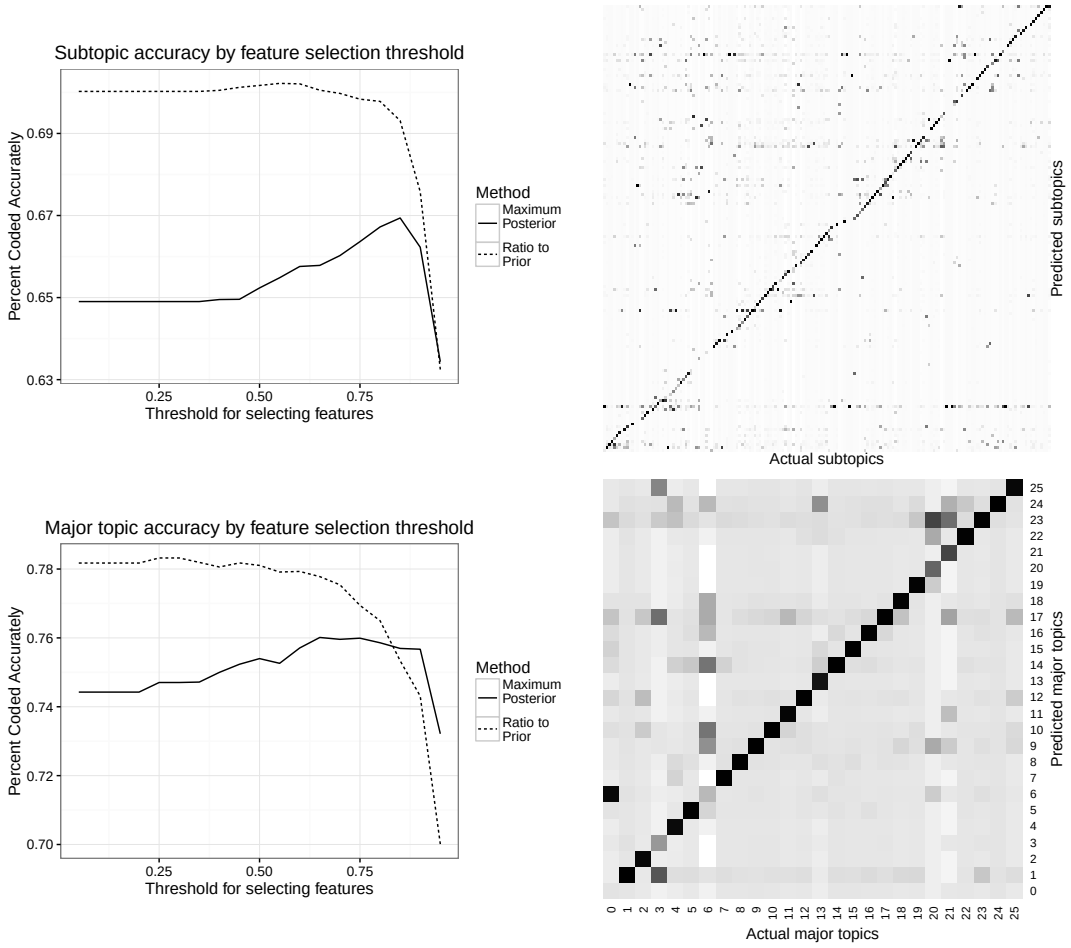

Figure 1: Accuracy of CAPCAS classifier

Note: Upper panels are for subtopics. Lower panels are for major topics. Left panels plot accuracy over proportion of training features removed. At each point, the algorithm removes all features below the respective quantile of mutual information. Right panels plot graphical confusion matrices of subtopic classification accuracy. Darker portions indicate more data. The dark diagonals indicate that accuracy is reasonably strong.

The right panels of figure 1 show graphical confusion matrices: computer classifications are plotted against the true classifications and the boxes are shaded by row to indicate where data are concentrated. Off-diagonal shaded areas indicate mistakes. The visualization is helpful in drawing attention to the computer's weaknesses, suggesting ways to refine features. For example, in the major topics plot (bottom right panel of figure 1), the non-policy related class 0 gets mistakenly classified as topic 6 and real instances of 6 are often mistakenly classified into 14, 10, and 9. These observations can guide feature refinements. In the subtopic plot (top right), the block patterns of misclassifications around the diagonal are instances of subtopics mistakenly classified into other subtopics within the same major topic.

The left panels of figure 1 plot classification accuracy against the proportion of training features retained for training the classifier. The threshold plotted on the x-axis indicates the quantile of mutual information below which features are dropped. Note that the optimal threshold varies between the two tests. For suptopics, the model benefits from using only around the top 40% of features by mutual information, while for major topics performs best when using nearly the top 75% of features.

As can be seen in the top left panel, overall test accuracy for subtopics reaches 70% in this demonstration, and matches the accuracy we record on our holdout data. Note that prior to the final collaborative coding step, accuracy rates of 80% were achieved, comparing favorably to intercoder-reliability rates between human coders. This test includes both human and machine-coded data, and therefore the results here suffer lower accuracy from propagating human and computer errors into the coding. Accuracy rates within many subtopics reach or exceed 95%. The two lines in the left-hand panels denote accuracy when using normal posterior probabilities to assign codings or when using regularized ratios of posterior to prior probabilities.

To demonstrate the performance of NB on more familiar data, we replicate the agenda coding of U.S. Congressional bills (Hillard, Purpura and Wilkerson, 2008). As data, we use the titles of the approximately 457,000 bills introduced in the U.S. Congress between

1947 and 2008 using data from the Congressional Bills Project (Adler and Wilkerson, 2015).[13] Features were refined by removing English stop words and stemming, but no expert knowledge was applied to further refine features. We divide the corpus randomly into three, training the algorithm on around 274,000 bills (around three-fifths of the data), testing on about 91,000, and measuring accuracy on a holdout data set of around 91,000.

Again, features with low values of mutual information are removed. Figure 2 reiterates that NB benefits from removing less informative features. When coding subtopics (top left panel), our results indicate the best subtopic performance is achieved when eliminating the lowest 89.7% of training features by mutual information. For major topics (bottom left panel), the algorithm only uses a little more than 50% of features to achieve the best result. Hillard, Purpura and Wilkerson (2008) record an accuracy rate, i.e. proportion of test documents correctly labeled, of around 71% using NB to classify subtopics. Our algorithm replicates this performance. For illustrative purposes, we again plot accuracy using both forms of posterior: regularized by ratio of posterior to prior and by maximum posterior probability.

Improving human computer collaboration

We recommend several tools for improving automated coding and present some examples here supporting our argument about the benefits of human and machine collaboration. Table 1 lists a sample of classes in the U.S. bills training data along with two measures of coding accuracy on the test data: the true frequency of classes in the test data, and the top five classes the algorithm mistakenly assigned to this class in the test data. These accuracy measures are the true positive rate and the positive predictive value. True positive rate is defined as the number of times the computer correctly predicted a class

[13] Views expressed are those of the authors and not those of the National Science Foundation.

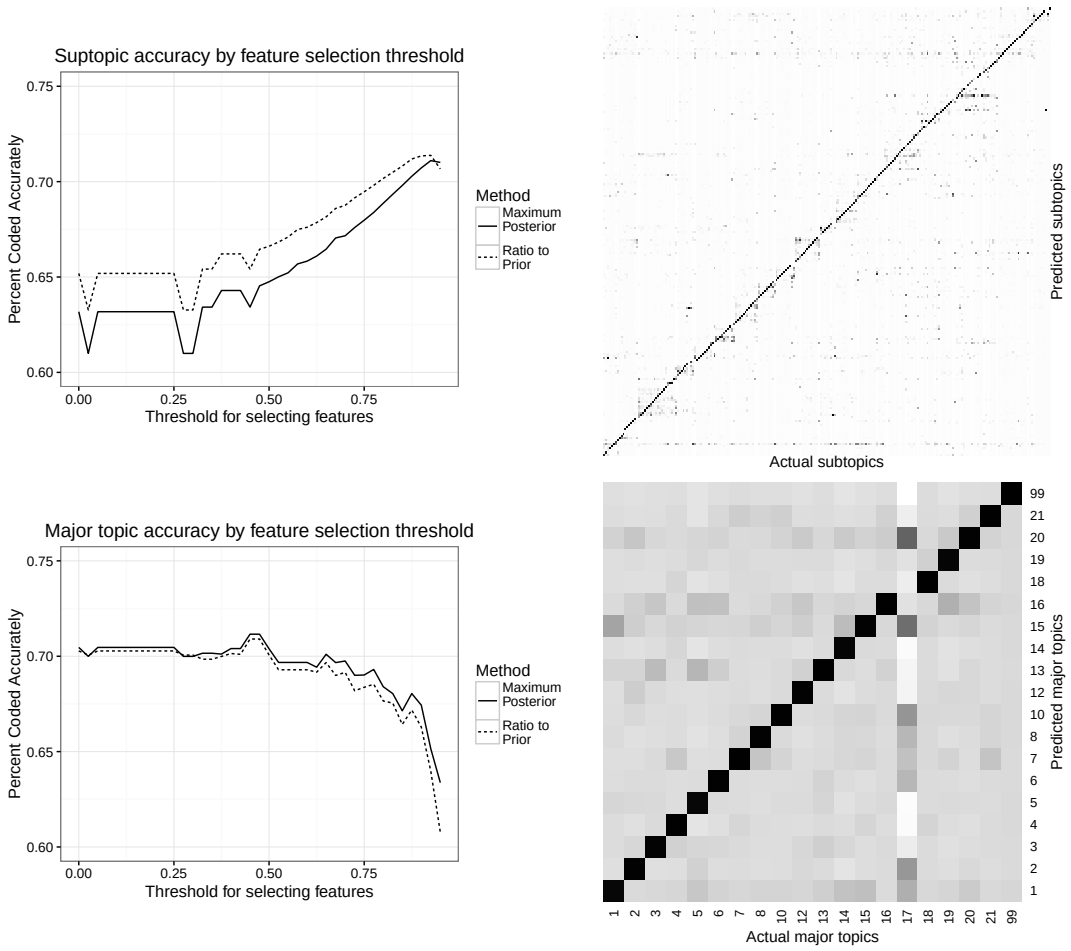

Figure 2: Accuracy of U.S. bills classifier

Note: Upper panels are for subtopics. Lower panels are for major topics. Left panels plot accuracy over proportion of training features removed. At each point, the algorithm removes all features below the respective quantile of mutual information. Right panels plot graphical confusion matrices of subtopic classification accuracy. Darker portions indicate more data. The dark diagonals indicate that accuracy is reasonably strong.

divided by the number of times the class truly appears in the test data. Positive predictive value is defined as the number of times the computer correctly predicted a class divided by the number of times the computer correctly predicted that class. Higher true positive rates indicate the algorithm is better at detecting the respective class. Higher positive predictive values indicate the algorithm is better at discriminating the class from others.

This tool is useful for allocating human coder time to checking machine codings. Classes with both high true positive rates and high positive predictive value are well-recognized by the algorithm and not often mistaken for other classes. These classes are strong candidates for being accepted as correct without human coder attention when coding virgin data. An example in table 1 would be class "1807" (Tariff & Imports)

Class	True Positive Rate	Positive Predictive Value	True Frequency	Top Mistaken Classes (1)	(2)	(3)	(4)	(5)
9999	0.968	0.993	39876	2006	900	1699	2103	1807
1807	0.873	0.945	7690	401	1800	1802	803	1604
2006	0.86	0.672	913	104	1699	1608	2002	1210
501	0.847	0.822	811	2004	1609	302	503	505
107	0.845	0.482	5887	503	2001	601	1521	2009
2030	0.845	0.925	556	2003	1609	2099	2006	2101
1521	0.819	0.708	1658	107	2002	503	1501	2007
1003	0.816	0.739	1961	107	2004	1210	1609	1002
801	0.811	0.746	613	704	107	800	806	802
529	0.419	0.842	229	530	504	503	505	1520
1201	0.412	0.497	1266	1209	1210	1204	2004	2000
1305	0.2	0.487	380	107	1699	502	601	1301
498	0.151	0.903	186	709	405	404	400	402
2100	0.087	0.219	644	2103	2104	2101	2000	2008
1099	0	0	30	1002	1007	1001	107	1006
1308	0	0	64	107	1208	322	1301	1304

Table 1: Accuracy by category, U.S. bills (excerpt)

or "9999." Note that "9999" is the catch-all class for bills not related to substantive policy. A high true positive rate and positive predictive value for this class is, therefore, important for classifier performance. Classes with only high positive predictive values are also strong candidates for being accepted without human coder intervention, because when the algorithm does predict the respective class it is often correct. An example in table 1 would be class "498" (Research & Development).

Classes with lower true positive rates indicate the analyst should refine the training features or apply human reviewer attention. Consider, for example, class "2100" (General public lands) in table 1. All five of the top mistaken classes are within the same major topic: 21 (Public lands). This indicates that the general subject of these items is clear, but specific features associated with the subtopic are less obvious. Directing attention to the specific words differentiating subtopic 2100 from others may help refine the training features to achieve greater accuracy. On the other hand, class "1305" (Social welfare volunteer organizations) in table 1 is often mistaken for classes outside its own major class. Though attention to the training features will likely improve accuracy, this class may require special attention from human reviewers.

Finally, the frequency with which classes appear as mistakes can be informative about where additional feature refinements and human coder effort will improve accuracy. For

example, class "107" (Tax code) frequently appears as a top mistaken class – 102 times in the full table. This accords with its lower positive predictive value of .482. This indicates the class is strongly associated with certain features appearing often in bill titles associated with other classes. This is corroborated by referring to our second assessment tool. Table 2 lists the top predicting words for each class. The word "now" is strongly associated with class "107," but this feature likely appears often in other classes. Removing "now" from the training data may improve the positive predictive value for class "107" and the overall predictive accuracy of the algorithm.[14]

	Top predicting words				
Class	(1)	(2)	(3)	(4)	(5)
101	cost-of-liv	inflat	automat	index	depreci
107	unmarri	old	marri	enjoy	now
110	stabil	inflat	price	april	prenotif
201	desegreg	racial	lynch	segreg	race
341	cigarett	tobacco	smoke	cigar	nicotin
400	farmland	farmer	stockyard	feed	packer
530	immigr	alien	visa	nonimmigr	deport
601	higher	cours	tuition	pursu	degre
705	clean	emiss	carbon	ozon	dioxid
710	sanctuari	coastal	wetland	spill	dump
803	gas	petroleum	crude	oil	pipelin
900	immigr	citizenship	alien	unus	visa
1001	mass	commut	transit	buse	instrument
1003	aviat	airport	airlin	airway	aeronaut
1203	narcot	marihuana	methamphetamin	substanc	cocain
1205	prison	parol	inmat	correct	bail
1401	neighborhood	grantsinaid	apart	block	repair
1501	deposit	depositori	bank	save	thrift
1522	copyright	patent	trademark	infring	invent
1524	tourism	travel	tourist	tour	hostel
1605	disarma	weapon	prolifer	nonprolifer	nuclear
1701	space	aeronaut	flight	astronaut	superson
1927	terrorist	terror	antihijack	murder	piraci

Table 2: Top words by category, U.S. bills (excerpt)

Table 2 lists a sample of subtopics in the training data along with the five training features most associated with each. Associations between features and classes are determined with respect to features. If, instead, we considered feature association with respect to classes, then certain very common features would be strongly associated with many categories. This would be uninformative about which features are most *uniquely* associ-

[14] We provide software for generating both of these assessment tools together with code for implementing our NB classifier.

ated with individual categories – i.e. which features discriminate most strongly among categories. As can be seen in this example, the features associated with particular classes often strongly indicate their subject matter. This is a good sign. This tool is useful for refining training features. Features that are clearly less informative about the class with which they are associated are good candidates for elimination from the training data or merging with a related term.

As examples, compare the features associated with classes "110" (Price control) and "1203" (Illegal drugs). Class "110" includes the name of a month: "april." Since months are not unambiguously associated with particular classes, this feature would be a strong candidate for deletion from the training data. On the other hand, all features listed as strongly associated with class "1203" give a clear sense of the class subject and therefore are strong candidates for retention.

Improving returns on research budgets

We return finally to our core motivation for adopting computer classification techniques: maximizing returns on research budgets. Several key decisions influence researchers' ability to get the most from their budgets. Here we provide a basic guide to some crucial decisions: setting the size of training sets; allocating human resources to coding versus review; and updating existing data with newly arrived data.

Training set size: As a general rule, classification accuracy improves with larger training sets. However, these accuracy boosts have diminishing returns. To illustrate, we rerun the U.S. bill titles classifier using differing sized training sets to code holdout data. Figure 3 plots accuracy by size of training set. At each size, we run the classifier ten times, randomly reselecting a training set of the respective size each time. We tune all classifiers on the same test data and record classification accuracy on the same holdout data. Boxplots summarize performance across all ten trials for each size.

Note two features of figure 3. First, performance varies more when training sets are

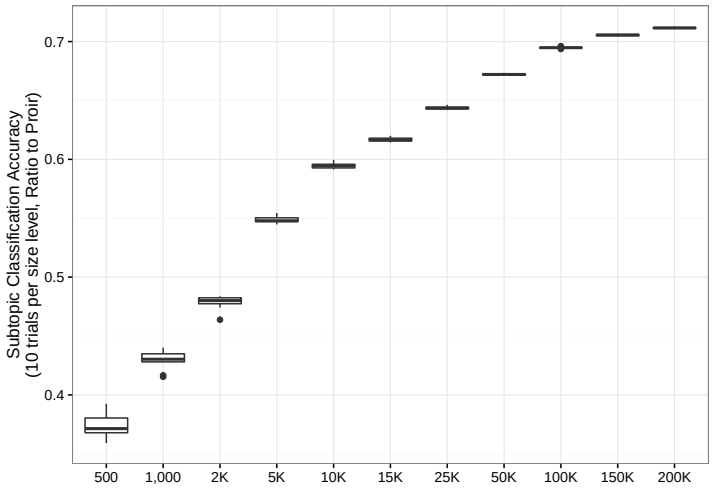

Figure 3: Accuracy of U.S. bills classifier by training set size
Note: Boxplots of classification accuracy for ten randomly selected training sets at each size level.

smaller. Second, accuracy improves more quickly when increasing the size of smaller training sets than when increasing the size of larger training sets. Thus, adding 5,000 training observations to a training set of size 1,000 improves accuracy much more than adding the same 5,000 training observations to a training set of size 50,000. Lastly, note this figure is only a guide – actual accuracy increases by training set size may vary depending on the data.

Allocating human coder effort: Using human coders to code virgin data or to review computer classification impacts the return on research budgets. Our experience indicates it is best to shift human effort from virgin coding to reviewing computer results as quickly as possible. Two pieces of empirical evidence are relevant. The first comes from the CAPCAS project budget, and the second from a lab experiment we conducted.

CAPCAS human coders were trained undergraduate research assistants. Based on internal calculations, coders received a contract in which they were paid for one hour of work for every 100 virgin items they coded. Actual efficiency varied. Early on, students reporting spending more than one hour to code 100 items, but after some experience all reported spending a little less than an hour per 100 items.

Coding commenced with producing a training set of 24,000 agenda items, at a cost of

240 coder hours. After this, coders were reassigned to validating computer codes and we agreed on a simple hourly wage for this work. The project paid a total of 440 coder hours to validate the remaining 176,000 items in our main data set. Thus, our data set of more than 200,000 agenda items was coded at a grand total cost of 680 coder hours – almost a 75% cost reduction from the originally budgeted 2,000 coder hours. Human coder self reports indicated the main efficiency gain came from the ability to group review material into large chunks of very similar agenda items, making the task of reviewing simpler than the task of coding virgin unsorted data.

To interrogate this apparent jump in efficiency, at the end of the project we paid the same research assistants to participate in a controlled experiment aimed at quantifying the efficiency gains from using human experts to validate computer codes relative to coding virgin data. We defined four treatment conditions. In each, coders received 100 agenda items to code. The control condition was a random sample of 100 unsorted virgin agenda items. Three test conditions involved some level of computer assistance: 1) computer classification hidden, but items are sorted by computer classification; 2) computer classification visible, but items appear in random order; 3) computer classification visible, and items are sorted by computer classification. All coders performed each task in random order.

Comparing the three test conditions to the control condition reveals advantages to computer coding. Relative to the control condition, we noted a statistically significant five percent improvement in average speed when coders had some computer assistance — an improvement likely to increase quite dramatically with the size of the task. We also found no evidence that the consistency or accuracy of human coding was lower when helped by the computer: human consistency and accuracy were equal between the control task and each test condition. This finding is important because it indicates human coders were not influenced by seeing the computer codes.

It is important to reiterate that this experiment used small samples and experienced coders. All coders had around eight months experience performing virgin coding and validating computer coding. Thus, we consider this a hard test for finding a significant

improvement in speed. Considering this evidence, and our actual experience, we consider it strongly advisable for researchers to use human coders to validate computer coding rather than to code virgin data whenever possible.

Efficiently updating existing data sets: Updating data cheaply is an important and promising application of computer classification. In short, we recommend researchers use temporally recent observations to code new data since language, issue salience, and other important features of policy can vary over time. Classification is generally most accurate when training using chronologically recent data.

We demonstrate with the U.S. bills data in figure 4. We classify all bill titles from years 2000 through 2014, one year at a time, taking training data from several time windows. We first classify each year's bills with training data consisting only of bills from the preceding five years, then we use training data consisting only of bills from between six and ten years earlier, etc. To avoid confounding the results with varying training set sizes, we train the classifier using 15,000 randomly sampled bill titles from the respective time windows. Classifiers were tuned on the bills omitted from the training data in each time window.

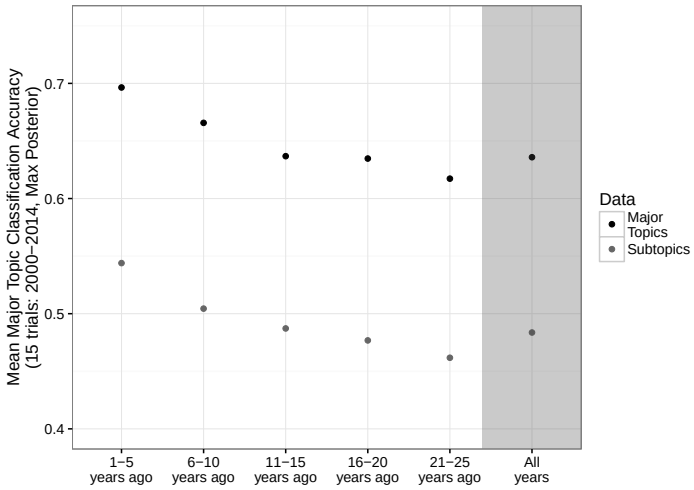

Figure 4: Classification accuracy by years used for training
Note: Points are mean accuracy across 15 trials. Each trial codes one year of data. Years span 2000 to 2014. Training sets each consist of 15,000 bill titles randomly sampled from the respective time window. Colored dots indicate results for major and minor topics.

The conclusion from figure 4 is that classification accuracy is greatest when using bills temporally closer to the new observations. This is what we should expect also because the factors driving the policy agenda change over time (Mettler, 2016). Note, further, that average classification accuracy on "future" bills is lower here than we would expect for training sets of 15,000 items, based on figure 3. This indicates that, although temporally recent training data is more informative than temporally distant data, it is still better to code a multi-year data set by creating a training set from a simple random sample of all data when there is no existing corpus of coded data. Classifying a new data set in chronological order is likely to yield decreased computer accuracy.

Conclusion and discussion

This paper argues that limits on budgets, time, and computing power require policy studies scholars to think outside the box about how to execute large data collection and coding projects. These have historically been a core data source for the field, and should continue to be. However, tightening research budgets and expanding government data make this an increasingly challenging task. We argue the way forward is to adopt a strategy for coding data that leverages human and computer strengths: humans supply some coding and expert knowledge to refine the computer classifications in order to build policy data quickly and accurately.

The findings reported here indicate that our workflow, combined with our implementation of NB, is one promising way of addressing these challenges. The CAPCAS project was able to produce a data set three times larger than originally envisioned and did so under budget and in far less time than was originally allocated for human coding. We consider this a strong recommendation of our method and a hopeful indication for future policy studies data collection projects.

It is important to stress that there is no obvious substitute for expert domain knowledge and careful human supervision of computer classification. Some scholars have recommended using machine translated data to conduct large-scale multilingual text modeling

by translating data from multiple languages into English.[15] In the context of policy data, this could imply it is possible to harness the vast amount of policy agendas data already coded to automatically code data cross-nationally. An additional analysis of legislative bills from eight countries, however, showed that this is likely not a viable strategy. More particularly, using the titles of legislative bills,[16] we trained our Naïve Bayes classifier using data from each country and in turn use each to classify the agenda subtopics of each other country.[17] The results from this exercise demonstrate that its success rate is currently quite low, topping out at 28.2% accuracy when using Danish legislative data to code Swiss laws. Accuracy increases slightly by refining features for high mutual information with subtopics, but never reaches even 40%.[18] The poor performance may be attributable to the quality of machine translation, variation in legal systems across countries, varying application of codes across countries, or varying wording used to describe subtopics across countries. Nevertheless, these results suggest that, at least with regard to coding agenda subtopics, there is not yet an easy substitute for applying expert knowledge to develop coding tools for use within single countries.

In conclusion, we would also reiterate that there are powerful alternative machine learning methods that agenda setting scholars may benefit from applying. We include a discussion of these in appendix E.

[15] See Lucas et al. (2015).

[16] Data available at `http://www.comparativeagendas.net/`.

[17] For this analysis, data from Denmark, Belgium, Spain, France, Hungary, and Switzerland were translated to English language. Data were translated using the Google Translate API, via the `translateR` package. Data from the U.S. and the U.K. were left in the original language. The resulting English text was stemmed, stopwords were removed, and all training features were used to train the classifier.

[18] See full results in appendix D

References

Adler, E. Scott and John D. Wilkerson. 2013. *Congress and the politics of problem solving.* Cambridge University Press.

Adler, E. Scott and John Wilkerson. 2015. "Congressional Bills Project: 1947-2008, NSF 00880066 and 00880061." http://congressionalbills.org/.

Baumgartner, Frank R. and Bryan D. Jones. 2016. "U.S. Policy Agendas Project. Support from National Science Foundation grant numbers SBR 9320922 and 0111611 and distributed through the Department of Government at the University of Texas at Austin." `http://www.comparativeagendas.net/us`.

Baumgartner, Frank R., Bryan D. Jones and Peter B. Mortensen. 2017. *Theories of the Policy Process.* 4th ed. Boulder, CO: Westview Press chapter Punctuated Equilibrium Theory: Explaining Stability and Change in Public Policymaking, pp. 55–101.

Burscher, Bjorn, Rens Vliegenthart and Claes H. De Vreese. 2015. "Using Supervised Machine Learning to Code Policy Issues: Can Classifiers Generalize across Contexts?" *The ANNALS of the American Academy of Political and Social Science* 659(1):122–131.

Carpenter, Daniel P. 2001. *The forging of bureaucratic autonomy: Networks, reputations and policy innovation in executive agencies, 1862–1928.* Princeton, NJ: Princeton University Press.

Chen, Jingnian, Houkuan Huang, Shengfeng Tian and Youli Qu. 2009. "Feature selection for text classification with Naïve Bayes." *Expert Systems with Applications* 36:5432–5435.

Cover, Thomas M. and Joy A. Thomas. 1991. *Elements of Information Theory.* John Wiley.

Frank, Eibe and Remco R. Bouckaert. 2006. *Knowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings.* Berlin, Heidelberg: Springer chapter Naive Bayes for Text Classification with Unbalanced Classes, pp. 503–510.

Green-Pedersen, Christoffer and Peter Bjerre Mortensen. 2013. "Danish Policy Agendas Project." http://www.agendasetting.dk.

Green-Pedersen, Christoffer and Stefaan Walgrave, eds. 2014. *Agenda setting, policies, and political systems: A comparative approach.* University of Chicago Press.

Grimmer, Justin and Brandon M. Stewart. 2013. "Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts." *Political Analysis* 21(3):267–297.

Hillard, Dustin, Stephen Purpura and John Wilkerson. 2008. "Computer-Assisted Topic Classification for Mixed-Methods Social Science Research." *Journal of Information Technology & Politics* 4(4):31–46.

Hovold, Johan. 2005. Naive bayes spam filtering using word-position-based attributes. In *CEAS 2005 - Second Conference on Email and Anti-Spam.* Palo Alto, California USA: .

Hsu, Chih-Wei, Chih-Chung Chang and Chih-Jen Lin. 2003. "A practical guide to support vector classification.".
URL: *http://www.csie.ntu.edu.tw/c̃jlin/papers/guide/guide.pdf*

Jakobsen, Mads Leth Felsager and Peter B. Mortensen. 2015. "How politics shapes the growth of rules." *Governance* 28(4):497–515.

Japkowicz, Nathalie and Shaju Stephen. 2002. "The class imbalance problem: A systematic study." *Intelligent Data Analysis* 6(5):429–449.

Jones, Bryan D. 2016. "The Comparative Policy Agendas Projects as measurement systems: response to Dowding, Hindmoor and Martin." *Journal of Public Policy* 36(1):31–46.

Jones, Bryan D., László Zalányi and Péter Érdi. 2014. "An integrated theory of budgetary politics and some empirical tests: the US national budget, 1791–2010." *American Journal of Political Science* 58(3):561–578.

Jurka, Timothy P., Loren Collingwood, Amber E. Boydstun, Emiliano Grossman and Wouter van Atteveldt. 2013. "RTextTools: A Supervised Learning Package for Text Classification." *The R Journal* 5(1):6–12.

Kibriya, Ashraf M., Eibe Frank, Bernhard Pfahringer and Geoffrey Holmes. 2005. *AI 2004: Advances in Artificial Intelligence: 17th Australian Joint Conference on Artificial Intelligence,*

Cairns, Australia, December 4-6, 2004. Proceedings. Berlin, Heidelberg: Springer chapter Multinomial Naive Bayes for Text Categorization Revisited, pp. 488–499.

Kremer, Jan, Kim Steenstrup Pedersen and Christian Igel. 2014. "Active learning with support vector machines." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4(4):313–326.

Lantz, Brett. 2015. *Machine Learning with R.* 2 ed. Packt Publishing.

Lewis, David. 2003. *Presidents and the politics of agency design.* Stanford, CA: Stanford University Press.

Lewis, David D. 1998. *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings.* Berlin, Heidelberg: Springer chapter Naive (Bayes) at forty: The independence assumption in information retrieval, pp. 4–15.

Lucas, Christopher, Richard A. Nielsen, Margaret E. Roberts, Brandon M. Stewart, Alex Storer and Dustin Tingley. 2015. "Computer-Assisted Text Analysis for Comparative Politics." *Political Analysis* 23(2):254–277.

March, James G., Martin Schulz and Xueguang Zhou. 2000. *The Dynamics of Rules: Change in Written Organizational Codes.* Stanford, CA: Stanford University Press.

McCallum, Andrew and Kamal Nigam. 1998. A Comparison of Event Models for Naive Bayes Text Classification. In *Learning for Text Categorization: Papers from the AAAI Workshop, AAAI Press.*

Metsis, Vangelis, Ion Androutsopoulos and Georgios Paliouras. 2006. Spam Filtering with Naive Bayes - Which Naive Bayes? In *CEAS 2006 - Third Conference on Email and Anti-Spam.* Mountain View, California USA: .

Mettler, Suzanne. 2016. "The Policyscape and the Challenges of Contemporary Politics to Policy Maintenance." *Perspectives on Politics* 14:369–390.

Munzert, Simon, Christian Rubba, Peter Meißner and Dominic Nyhuis. 2015. *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining.* John Wiley & Sons Ltd.

O'Neil, Cathy and Rachel Schutt. 2013. *Doing Data Science: Straight Talk from the Frontline.* O'Reilly Media.

Ostrom, Elinor with Cox, Michael and Edella Schlager. 2014. *As Assessment of the Institutional Analysis and Development Framework and Introduction of the Social-Ecological Systems Framework.* Boulder, CO: Westview Press.

Plazek, David J. and Alan Steinberg. 2013. "Political Science Funding Black Out in North America? Trends in Funding Should not be Ignored." *PS: Political Science & Politics* 46(3):599–604.

Schneider, Karl-Michael. 2005. *Computational Linguistics and Intelligent Text Processing: 6th International Conference, CICLing 2005, Mexico City, Mexico, February 13-19, 2005. Proceedings.* Berlin, Heidelberg: Springer chapter Techniques for Improving the Performance of Naive Bayes for Text Classification, pp. 682–693.

Schohn, Greg and David Cohn. 2000. "Less is more: Active learning with support vector machines." *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)* pp. 839–846.

Schulz, Martin. 1998. "Limits to Bureaucratic Growth: The Density Dependence of Organizational Rule Births." *Administrative Science Quarterly* 43(4):845–876.

Thesen, Gunnar. 2013. "When good news is scarce and bad news is good: Government responsibilities and opposition possibilities in political agendasetting." *European Journal of Political Research* 52(3):364–389.

Tong, Simon and Daphne Koller. 2001. "Support Vector Machine Active Learning with Applications to Text Classification." *Journal of Machine Learning Research* 2:45–66.

Train, Kenneth. 2009. *Discrete Choice Methods with Simulation.* 2 ed. Cambridge University Press.

True, James. 2009. "Historical Budget Records Converted to the present functional categorization with actual results for FY 1947-2008.".
URL: *http://comparativeagendas.s3.amazonaws.com/adhocfiles/Budget_Codebook_Comprehensive.pdf*

van Witteloostuijn, Arjen and Gjalt de Jong. 2007. "The Evolution of Higher Education Rules: Evidence for an Ecology of Law." *International Review of Administrative Sciences* 73(2):235–255.

Vlachos, Andreas. 2004. Active Learning with Support Vector Machines PhD thesis The University of Edinburgh.

Vliegenthart, Rens, Stefaan Walgrave, Frank R. Baumgartner, Shaun Bevan, Christian Breunig, Sylvain Brouard, Laura Chaqués Bonafont, Emiliano Grossman, Will Jennings, Peter B. Mortensen, Anna M. Palau, Pascal Sciarini and Anke Tresch. 2016. "Do the media set the parliamentary agenda? A comparative study in seven countries." *European Journal of Political Research* 55(2):283–301.

Workman, Samuel. 2015. *The Dynamics of Bureaucracy in the U.S. Government: How Congress and Federal Agencies Process Information and Solve Problems.* Cambridge University Press.

Yesilkagit, Kutsal and Jørgen G. Christensen. 2010. "Institutional design and formal autonomy: Political versus historical and cultural explanations." *Journal of Public Administration Research and Theory* 20(1):53–74.

Yu, Bei, Stefan Kaufmann and Daniel Diermeier. 2008. "Classifying Party Affiliation from Political Speech." *Journal of Information Technology & Politics* 5(1):33–48.

Zhou, Xueguang. 1993. "The Dynamics of Organizational rules." *American Journal of Sociology* 98(5):1134–1166.

Appendix
Not for Print Publication

A Appendix: Multiclass Naïve Bayes with Bernoulli features

Documents are represented as vectors of binary indicators for the appearance or absence of features: $d_i \equiv \{f_1, f_2, \ldots, f_j\}$. The probability of observing feature j (f_j), in a document belonging to class c is distributed Bernoulli: $\hat{\theta}_{jc}^{f_j}(1-\hat{\theta}_{jc})^{(1-f_j)}$. The $\hat{\theta}_{jc}$ values are prior probabilities of observing feature j in a document from class c. The hat over it indicates it is estimated from the training data. Likewise, $\hat{\theta}_c$, is the prior probability of observing class c, unconditional on its features. Both estimates are based on the empirical frequency of classes and features in the training documents. By the independence assumption, the prior probability of observing document i in class c is thus:

$$P(d_i|c) = \prod_j \hat{\theta}_{jc}^{f_{ij}}(1-\hat{\theta}_{jc})^{(1-f_{ij})}$$

Our outcome of interest is the probability a new document falls into class c, given its features. We calculate this probability for every possible class. To do so, we consider the (log) odds of observing each class, c, relative to one reference class, $\emptyset$. We drop the unconditional probability of observing document i, as it does not affect this ratio:

$$\begin{aligned}
\frac{P(c|d_i)}{P(\emptyset|d_i)} &= \frac{P(d_i|c)P(c)}{P(d_i|\emptyset)P(\emptyset)} \\
\frac{P(c|d_i)}{P(\emptyset|d_i)} &= \frac{\prod_j \hat{\theta}_{jc}^{f_{ij}}(1-\hat{\theta}_{jc})^{(1-f_{ij})} \times \hat{\theta}_c}{\prod_j \hat{\theta}_{j\emptyset}^{f_{ij}}(1-\hat{\theta}_{j\emptyset})^{(1-f_{ij})} \times \hat{\theta}_\emptyset} \\
\log\left(\frac{P(c|d_i)}{P(\emptyset|d_i)}\right) &= \sum_j \left[f_{ij} \times \left(\log\left(\frac{\hat{\theta}_{jc}(1-\hat{\theta}_{j\emptyset})}{\hat{\theta}_{j\emptyset}(1-\hat{\theta}_{jc})}\right) + \log\left(\frac{1-\hat{\theta}_{jc}}{1-\hat{\theta}_{j\emptyset}}\right)\right)\right] + \log\left(\frac{\hat{\theta}_c}{\hat{\theta}_\emptyset}\right)
\end{aligned}$$

The training step involves calculating all priors ($\hat{\theta}_c$ and $\hat{\theta}_{jc}$) from the training data. The classification step, involves using the expression above to calculate the posterior probability each classified document falls into each class.

B Appendix: Experimental results on human reviewer speed and accuracy

This appendix presents the results of our experiment to check the speed and accuracy implications of our approach to using human coders to review computer codings. Participants were student workers employed as human coders on the CAPCAS project. All had more than six months experience coding local government policy agendas, both with and without computer assistance.

The experiment consisted of 12 trials, with the first two discarded as warm-up rounds. Each trial presented the student with a data set of 100 randomly selected local government agenda items. The student applied codes to these agendas as quickly as possible. Each data set was randomly sorted into one of four treatment categories. The first category, the control or baseline, listed agenda items in random order with no computer coding visible. The second treatment category listed agenda items ordered by their computer codes, but the computer codes themselves were hidden from the participant. The third category listed agenda items in random order with a computer code visible next to each. The fourth and final category listed agenda items with a computer code visible next to each and items were sorted by computer coding – putting agendas with similar content near each other.

	Model 1
T2: Ordered	-2.75^{***}
	(0.14)
T3: Computer coded	-1.35^{***}
	(0.10)
T4: Ordered & computer coded	-1.54^{***}
	(0.10)
FE Coder 1	3.25^{***}
	(0.09)
FE Coder 2	6.32^{***}
	(0.09)
FE Coder 3	9.56^{***}
	(0.09)
Task number (time trend)	-0.30
	(0.16)
Constant	13.47^{***}
	(1.32)
Num. obs.	4000
Num. groups: Data set	10
Var: Data set (Intercept)	2.20
Var: Residual	3.99

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table A.1: Linear model of time spent coding, random effects by data set

Table A.1 analyses the time participants took to complete each trial. The outcome

variable is the participant's time, in minutes, to complete all 100 codings. The explanatory variables of interest are the three treament categories: T2, T3, and T4. Each of these three is an indicator variable taking on a one for observations under the respective treatment condition and a zero otherwise. The control category (T1) is the baseline category and is, therefore, excluded from models. Controls included are fixed effects for coders, a time trend for the number of the trial, and random effects by data set.

The results clearly indicate that all three treatment categories are associated with reduced human coder time, as we have argued. Notably, the treatment associated with the greatest speed increases was the use of ordered data sets without computer codes visible (T2). We can only speculate on the reasons for this difference. Having trained the coders to second-guess the computer-generated codings, we suspect that viewing the computer codes added an additional layer of complexity to the task of coding agenda items.

Importantly, as table A.2 shows, none of the treatment or control variables show a statistically significant relationship to the accuracy of the coding. The outcome variable here is a binary indicator for the accuracy of each coded item: correctly coded items take on the value of one and incorrect items are coded zero. Items were considered correct if the experiment participant's classification agreed with that of the authors. Importantly, not only are the coefficient estimates on all three treatments in the model statistically indistinguishable from zero, but they also represent effect sizes so small as to be substantively meaningless. To illustrate, the model in table A.2 predicts that the baseline participant assigned to the control treatment in round five has about a 74.4% probability of coding a given agenda item correctly (ignoring random effects for the moment). If we calculate predicted accuracy for the same participant under treatment two, the point prediction is a 77% probability of a correct coding. For treatment three the point prediction is 75.7% and for treatment for it is 75.4%. Thus, table A.2 tells us that there is likely no statistical relationship between accuracy and any treatment condition, but our best guess is that if there is a relationship it is *positive* but small enough to be meaningless.

Taken together, these results support our conclusion that human coder time is best invested in checking computer codes and that trained coders should not experience reduced accuracy in the process.

	Model 1
T2: Ordered	0.14
	(0.16)
T3: Computer coded	0.07
	(0.10)
T4: Ordered & computer coded	0.05
	(0.10)
FE Coder 1	0.19
	(0.11)
FE Coder 2	0.01
	(0.10)
FE Coder 3	−0.16
	(0.10)
Task number (time trend)	−0.00
	(0.02)
Constant	1.08***
	(0.15)
Num. obs.	4000
Num. groups: Data set	10
Var: Data set (Intercept)	0.00

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table A.2: Logit model of coding accuracy, random effects by data set

C Appendix: Mutual information and optimal tuning

We calculate mutual information using the measurement presented in McCallum and Nigam (1998), originally from Cover and Thomas (1991). This has the following form. Random variable C represents classes and random variable W_t takes on values $f_t \in \{0, 1\}$ indicating the absence or presence of word w_t in a document (0 = word is absent, 1 = word is present).

Quoting McCallum and Nigam (1998): "Average mutual information is the difference between the entropy of the class variable, $H(C)$, and the entropy of the class variable conditioned on the absence or presence of the word, $H(C|W_t)$."

Formally, they express this as:

$$\begin{aligned} I(C; W_t) &= H(C) - H(C|W_t) \\ &= -\sum_{c \in C} P(c)\log(P(c)) + \sum_{f_t \in \{0,1\}} P(f_t) \sum_{c \in C} P(c|f_t)\log(P(c|f_t)) \\ &= \sum_{c \in C} \sum_{f_t \in \{0,1\}} P(c, f_t)\log\left(\frac{P(c, f_t)}{P(c)P(f_t)}\right) \end{aligned}$$

The values $P(c)$, $P(f_t)$, and $P(c, f_t)$ are prior probabilities of seeing class c, seeing term (or word) t, and seeing both class c and term t. These are calculated by tabulating over the training data. The probability of observing class c is simply the frequency of class c in the training data divided by the total number of training observations. The probability of observing term t is simply the frequency of term t divided by the number of training observations. The joint probability of observing c and t is calculated by summing up the times a document in class c includes term t and then dividing by the number of training observations.

Summing over categories then yields the average mutual information for each term in the training data. Since Naïve Bayes works best when uninformative terms are removed from the data, eliminating terms with low mutual information can be a helpful step in boosting classification accuracy. Dropping features with low information is part of the process of "tuning" the model for getting better classification.

Tuning should be done systematically. Therefore, in all of our applications we select an optimum threshold for mutual information by training a model and testing its performance repeatedly. This iterative process is a necessary step in any application of machine learning, so we briefly discuss two standard approaches to this: holdout and cross-validation (see, for an accessible discussion Lantz, 2015, pp. 336-343).

In holdout validation, the analyst randomly splits training data into three parts: training, test, and holdout data. Tuning choices are optimized by repeatedly training the computer using the training subset under different conditions, then testing accuracy by classifying the test data under each setting. Once optimal accuracy is achieved, accuracy is checked by classifying the holdout subset under the optimal settings. The accuracy achieved on the holdout data estimates the classifier's performance on virgin data. Cross-validation, often called k-fold cross-validation, begins by randomly splitting the full training data into k subsets, or "folds." The classifier is iteratively trained using

$k-1$ subsets and tested on each held out subset. This process repeats k times, until each subset is used to check classifier accuracy. Overall accuracy is measured by averaging performance across the k folds. Tuning choices are optimized by repeating k-fold cross-validation under different tuning settings until reaching peak accuracy.

A threshold for feature inclusion is some minimum value of mutual information — terms with mutual information values less than or equal to this threshold are excluded from the training step. In our tuning, we select many different thresholds until finding the optimal setting for classification success on a test data set.

The same procedure applies to all other tuning choices. For example, how does one decide between regularized or simple posterior probabilities? Should punctuation be deleted or kept? Should a particular feature be removed from the training data? All of these need systematic validation.

Before moving on to computer-only coding (stage three) from collaborative coding (stage two), we recommend analysts always do a final thorough tuning by validating modeling choices using either holdout- or cross-validation.

D Appendix: Cross-national coding of CAP bill titles

Given the notable scope for applying computer codings to policy agendas data, it is reasonable to consider whether it might be possible to harness the vast amount of policy agendas data already coded to automatically code data cross-nationally. After all, Lucas et al. (2015) recommend using machine translated data to conduct large-scale multilingual text modeling by translating data from multiple languages into English. We put this possibility to the test here. Using the titles of legislative bills in eight countries,[19] we train our Naïve Bayes classifier using data from each country and in turn use each to classify the agenda subtopics of each other country. Results on accuracy from each trial are reported in table A.3.

		Test Data							
		DK	BE	ES	FR	HU	CH	US	UK
Training Data	Denmark		0.239	0.189	0.134	0.242	0.281	0.117	0.243
	Belgium	0.162		0.147	0.132	0.172	0.115	0.058	0.145
	Spain	0.181	0.160		0.093	0.190	0.146	0.085	0.118
	France	0.068	0.114	0.059		0.101	0.063	0.024	0.047
	Hungary	0.209	0.227	0.242	0.162		0.198	0.114	0.160
	Switzerland	0.219	0.151	0.196	0.098	0.201		0.078	0.145
	United States	0.240	0.184	0.125	0.148	0.201	0.225		0.212
	United Kingdom	0.260	0.247	0.165	0.103	0.186	0.220	0.124	

Table A.3: Machine coding accuracy on translated data

For this analysis, data from Denmark, Belgium, Spain, France, Hungary, and Switzerland were translated to English language.[20] Data from the U.S. and the U.K. were left in the original language. The resulting text was stemmed and all training features presented in the respective test data were used to train the classifier. The results demonstrate that the success rate for this is currently quite low, topping out at 28.2% accuracy when using Danish legislative data to code Swiss laws. Accuracy increases slightly by refining features for high mutual information with subtopics, but never reaches even 40%.

[19] Data available at `http://www.comparativeagendas.net/`.

[20] Data were translated using the Google Translate API, via the `translateR` package.

E Appendix: Advanced coding methods

There are a variety of powerful alternative machine learning methods that policy and public administration scholars may benefit from applying. The most promising is likely "active learning" combined with support vector machines (SVM). SVM represents documents as points in multidimensional space – one dimension per feature in the data. In the barest terms, classification is done by drawing surfaces (hyperplanes) that divide this multidimensional space separating, as homogenously as possible, classes from one another.[21] The algorithm finds hyperplanes with the largest margins – that is, with the maximum distance between the classes they divide.

Active learning is a principled method for selecting exactly which documents to ask human coders to classify. It proceeds from the assumption that coding documents is costly – an appropriate one for policy studies. The goal is to start with a small training set, and then to add new training data as efficiently as possible so classification accuracy improves quickly with minimal investments in human coding. Generally, the work flow proceeds similarly to our proposal. It begins with human-coded training data to classify all virgin data. The difference is that, with active learning, information from the classification step is used to select *which* virgin observations a human coder should classify in order to achieve the biggest accuracy boost.

Criteria for selecting new virgin observations for human coding depend on the classification method. For multiclass SVM, these include selecting observations on or near the margins around hyperplanes, observations on or near important areas of the margins, batches of observations with the most diversity in these measures, or various other rules.[22]

Combining SVM with active learning is among the most sophisticated approaches to human-computer collaborative classification currently in use, and it can perform demonstrably better than Naïve Bayes. It is worth briefly considering the drawbacks to utilizing it, however. The first is the obvious technical hurdles facing efforts to code policy data using more sophisticated methods. These are not insurmountable: multiclass SVM can be implemented in `R`, Python, and other statistical software packages. More important is the black box nature of SVM output. Due to the algorithm's complexity, tracing *how* SVM arrives at the classifications it produces is not feasible. As long as it predicts accurately, this is not necessarily a problem for the scientific study of policy processes. However, SVM and other methods will perform more quickly and accurately given data with more informative features – i.e. they still benefit from expert human tuning. Given the transparent and interpretable results Naïve Bayes produces, we recommend policy scholars planning to apply more sophisticated methods still begin with our method as a means of pruning features and learning about their data.

[21] See, for example, Lantz (2015, pp. 239-248) or the useful introduction by Hsu, Chang and Lin (2003) and documentation for their `libsvm` software.

[22] See, for examples, Vlachos (2004), Schohn and Cohn (2000), Tong and Koller (2001), and (Kremer, Steenstrup Pedersen and Igel, 2014).