Automated Coding of Open-Ended Survey Responses

Dallas Card and Noah A. Smith

October 2015

Abstract

Background: Manual coding of open-ended survey responses is time-consuming, expensive, and error-prone. However, this task can be recast as a multi-label text classification problem, to which natural language processing and machine learning techniques can be applied.
Aim: The aim of this work is to assess the suitability of various machine learning techniques to the problem of automated coding of open-ended survey responses, and to explore the use of techniques to automate the search for an appropriate model.
Data: The data for this project consist of approximately 28,500 manually-coded responses by 2,000 respondents to open-ended survey questions from the 2008 American National Election Studies survey.
Methods: We compare the application of two types of machine learning methods to this task: L1-regularized logistic regression, and recurrent neural networks. In an effort to provide a fair comparison, while minimizing bias in favor of novelty, we make use of Bayesian optimization to perform an automated search over the configuration space of these two types of models, and make use of a reusable holdout in an attempt to prevent overfitting.
Results: In the case of a traditional logistic regression model, we find that Bayesian optimization is an effective way to do feature selection and hyperparameter tuning. For recurrent neural networks, which involve a much larger space of possible configurations, we have been unable to produce a consistent improvement over the baseline in performance on this data.
Conclusions: Although we do not presently recommend recurrent neural networks as a effective or easy-to-use approach to this kind of limited-data multi-label classification problem, we believe that with further work, recurrent neural networks do have advantages which can be realized through an automated model selection process.

1 Introduction

Open-ended surveys offer social scientists the opportunity to learn about public opinion in respondents' own words. They allow discovery of opinions that might be unexpected when preparing, for example, a list of answer choices in a closed-ended survey. Open-ended surveys, however, have a high cost in analyzing responses.

The survey data we consider here is from the American National Election Studies (ANES), which seeks to ascertain the American public's knowledge of and attitudes about American politics. As with most data from open-ended survey questions, each of the ANES survey responses has been manually classified using predefined sets of labels, or "codes". This process, known as "coding", has a number of disadvantages, all of which might be overcome through automation:

- It requires human coders, who in turn require training, pay, and time to complete the task.
- Human coders can lack internal and group consistency, and they make errors. Considerable overhead is often required to check agreement among coders, synchronize them on coding conventions, and correct mistakes.

- Traditional coding methodology offers no measurement of coder confidence in making a judgment and no way to record rationales for judgments.

Automated coding could, potentially, offer a faster, lower-cost, and more consistent alternative, with additional information about confidence and rationales.

In this report, we explore and evaluate methods for automated coding of open-ended survey responses. Our primary focus is on comparing methods in terms of accuracy (measured against human coders), interpretability, and computational demands. A secondary focus is on describing the techniques we consider, which are all instances of general-purpose statistical text categorization models. We begin by reviewing the ANES dataset used in our experiments (Section 2), then present the methods (Section 3), and finally discuss our experiments (Section 4).

2 Data

Our data is a collection of open-ended questions and responses from the 2008 ANES survey. The responses are encoded as short textual summaries written by the interviewers, during the interviews. Prior to this work, the responses were manually coded by professionals (contracted by the designers of the survey) into a set of predefined codes for each question. To our knowledge, the iterative coding process followed standard, community-accepted procedures to obtain high intercoder agreement.

Full audio of conversations with survey respondents is recorded during ANES interviews, but—like the human coders—we do not make use of the audio recordings. This could potentially be interesting in future work, though we expect large amounts of irrelevant content in the audio stream. A secondary consideration is preserving respondents' privacy when sharing audio data, which is arguably less anonymous than textual summaries.

The 2008 ANES included twenty-one open-ended questions asked of over 2,000 respondents, dealing with the respondents' knowledge of and attitudes towards political issues, parties, and individuals. (For a complete list of questions, see Appendix A.) The number of codes per question ranges from 12 to 74, though some are never used in this dataset. Although some of these codes are mutually exclusive, in practice there was no restriction as to which or how many of the codes associated with a question a coder might give to any particular response.

For some questions, the available codes are quite carefully delineated (distinguishing, for example, between "Vice President" and "Vice President of the United States"). Others had more topical or thematically defined responses, such as "Defense spending," "Iraq," and "Terrorism." Because these were open-ended questions, there is the possibility of a response which cannot be well categorized given the predefined coding scheme, so categories such as "Other" are used to account for this possibility. Two additional responses common to all questions were "I don't know" and "Refuse to answer."

The questions were presented to respondents in a set order in a limited amount of time, thus not every question was asked of every respondent, and the number of responses per question thus varied from approximately 100 to 2,100, with a total of 28,500 individual responses.

Because of the nature of this text (generated quickly by interviewers attempting to summarize responses), it tends to be quite noisy, with frequent and varied misspellings and lack of grammatical structure. In addition, interviewers made use of shorthand, e.g., "dk" for "don't know", and "//" or "\\" to separate parts of a response. Unfortunately, these conventions were not consistent across interviewers. Table 1 gives some examples of unedited survey responses to a selection of these questions.

In what follows, we will use the term *question* to refer to one of the twenty-one survey questions (e.g., "Why do you think John McCain lost the Presidential election?"); *response* refers to the text

Question	Responses
Who is Dick Cheney?	Vice President vp vice prcident lobbyist/state senater I DON'T LIKE HIM// SKIP IT// know him but dont know what he doed vp, he's a jerk, dont give him a gun
Why might you vote for Barack Obama?	democrat good speeker he wants 2 end the war his understanding of the conomy for working class, support women, health care issues, democratic He is a black man. First black president. It going to go down in history. I agree with his views on several things, he's very experienced and knows what he's talking about/his views on the war, gay rights, abortion, the economy right now, health benefits, taxes and education/i just think he seems like a candidate who can connect with the people and be a good representation of the nation's population/
Why might you vote against Barack Obama?	his inexperience his views on abortion his muslim backround his issues on abortion and gay marriage; some people say that he is the anti christ; some people say he might be a terrorist his past ties w/ bill ayres his connection to the rev wright and the priest fleder. he got a loan from country wide and got a huge break and as far as i'm concerned this man is the devil he's alittle too liberal, i wanted hillary

Table 1: Examples of survey responses. The questions have been paraphrased, but the responses have been left uncorrected.

written to summarize one respondent's answer to one question. A *label* is an element of the set of a question's predefined categories that can be assigned to a response. For each response, each label is associated with a *value* of 1 (if the label was assigned to the response) or 0 (if it wasn't). The (complete) *code* for a response should be understood as a binary vector in which each dimension is associated with a label. We will use the term *prediction* to refer to an automatic coding method's code vector for a particular response.

3 Methods

We view the task of automatic survey response coding as a multi-label classification problem. In particular, for each survey question, determining which codes should be assigned to a particular response is a list of yes or no decisions. Thus, we can train classifiers to predict, for a new response, whether each code for the appropriate question should be assigned to the response or not. The predicted list of yes/no decisions provides a binary code vector.

In order to take advantage of the fact that some questions have identical label sets, we combine questions that share a label set. This results in ten groups of questions, as described in Appendix A. We will report, for each group, a single performance score, which can be understood as an average of per-response F_1 scores. Concretely, let $\mathbf{z}_i^*$ be the binary vector code assigned by human coders to the ith response in a group, and let $\hat{\mathbf{z}}_i$ be the binary vector code predicted by an automatic method for the ith response. Then we report:

$$\frac{1}{I}\sum_{i=1}^{I}\frac{2\mathbf{z}_i^{*\top}\hat{\mathbf{z}}_i}{\|\mathbf{z}_i^*\|_1+\|\hat{\mathbf{z}}_i\|_1} \tag{1}$$

where I is the number of responses to the union of questions with the same set of labels.

As usual in machine learning experiments, we split the data into training, validation, and test datasets. The training dataset is used to select model parameters for any given choice of predictive model and its hyperparameters. The validation dataset is used to estimate the performance of a trained model separately from the training dataset, so that we can compare different hyperparameter settings. To get a clean estimate of performance even after multiple rounds of consultation with the validation dataset, we turn at the end to the test dataset. In the following, the test dataset is a random 10% of the responses for each question, and the remainder of the data is divided into five equal-sized folds, which can alternately be used for training or validation data.

We consider two broad classes of approaches to this classification task: linear models and recurrent neural networks. Below, we give a basic explanation of each (Sections 3.1 and 3.2), though the differences are quickly summarized:

- Linear models are transparently interpretable, making use of words (and perhaps phrases) as features whose impact on a prediction can be quantified by inspection of the model parameters. Neural networks are, at present, less straightforward to interpret.
- Neural networks provide a much more expressive space from which to select hypotheses.[1] This might allow greater accuracy at making predictions, but it also means that far more training examples are needed to learn to generalize well.
- The computational task of training a linear model is very well understood: it is a convex optimization problem. This is not the case for neural networks, which tend to be much more

[1]The term *hypothesis* in machine learning refers to a function from inputs (here, survey response summaries) to outputs (here, sets of labels).

expensive to train. The convergence properties of neural network training algorithms are not well understood theoretically, though there is a rich literature of empirical findings.

Of course, many more options are available for classification. We have chosen these two for the simple reason that one (linear modeling) is a familiar and effective baseline in text classification tasks (Wang and Manning, 2012), and that the other has recently received a flood of attention in the computer science community. Our expectation in the long term is that the best methods to be developed will be hybrids that draw from the ideas underlying both of these families of techniques, and others as well.

For both kinds of models, there are a number of design decisions to make and hyperparameters to select. These include numerical choices like the strength of a regularization penalty or the length of a neural network "hidden" vector, as well as discrete choices like whether to include bigram features in a linear model or which type of nonlinear function to use in a neural network. (We call a full configuration of choices a *setting*.) We therefore rely on *Bayesian optimization* to resolve these choices in reasonable ways, under computational budget constraints (Bergstra et al., 2011; Snoek et al., 2012). In Bayesian optimization, a sequence of settings is explored, with each setting chosen based on the observed performance of settings in previous rounds, estimated on a validation dataset. Because this method risks overfitting to the validation set, we use a *reusable holdout* technique (Dwork et al., 2015). Bayesian optimization and the reusable holdout are discussed in greater detail in Appendices B and C, respectively.

3.1 Linear Models (Logistic Regression)

We consider logistic regression (LR), a family of linear models with a probabilistic semantics. The prediction rule LR uses to decide whether to assign label ℓ to textual response i is:

$$\hat{z}_{i,\ell} = \begin{cases} 1 & \text{if } \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\mathbf{x}_i$ is a feature representation of the textual response and $\langle \beta_0, \boldsymbol{\beta} \rangle$ is a vector of label-specific coefficients estimated through the training procedure to maximize, for each label ℓ:

$$\sum_i \log p_{\boldsymbol{\beta}}(Z = z^*_{i,\ell} \mid \mathbf{x}_i) - \lambda R(\boldsymbol{\beta}) \tag{3}$$

where i here ranges over responses in the training dataset, R is a regularization penalty, and λ is a hyperparameter. The probability of a label is given by the classic LR parametric form:

$$p_{\boldsymbol{\beta}}(Z = 1 \mid \mathbf{x}_i) = \left(1 + \exp(-\beta_0 - \boldsymbol{\beta}^\top \mathbf{x}_i)\right)^{-1} \tag{4}$$

How well LR works depends, of course, on the features. The usual starting point for defining a feature vector for text classification is a histogram of word occurrences in an instance, so that each word type gets a coefficient. We also consider consecutive two-word sequences (bigrams), a feature for the question name, and conjunctions of words with the question type, as well as abstractions from words (part-of-speech tags, word Brown clusters (Brown et al., 1992),[2] and named entities). The number of instances of each type of feature are shown in Table 2.

[2]Brown clustering assumes a hierarchical language model which models the probability of a word given its cluster, and the probability of transitioning to each cluster given the cluster of the previous token. Each word type is assigned to a single cluster, with the assignments determined by greedily merging clusters to maximize the likelihood of a corpus of text until the desired number of clusters is reached.

Feature type	Number of instantiated features
Basic n-grams:	
Unigrams	11,467
Bigrams	65,460
Conjoined n-grams:	
Democrat vs. Republican	7,673
Likes vs. Dislikes	11,906
Clinton vs. Obama	4,982
McCain vs. Obama	8,216
Personal vs. Political	5,401
Other	
Brown clusters	200
Part-of-speech tags	45
Named entities	13
Question name	16

Table 2: Features considered for logistic regression. The number of instantiated features is given without thresholding based on the number of times a feature occurs in the data.

The above discussion gives a sense of the kind of information that our LR model can use to code a response, but there are many more decisions to be made to precisely specify the features and the training method. These decisions, we argue, are not intuitive or interesting, in the sense that an understanding of the domain will not shed much light on the best choices. They include: whether to convert characters to lower case (we do); where exactly to split responses into tokens (we use whitespace and slashes, and split off punctuation); whether to convert counts into binary values; whether to throw out features that occur too few times in the training data (and if so, what frequency threshold); how strongly to regularize (i.e., the value of λ in Equation 3); and whether to include each type of feature or ignore it. Because we do not have the resources for an exhaustive exploration of all of these options, we will report the results of two procedures:

1. A baseline in which we include only binarized unigrams, a classic setup that is cheap to implement, and tune the regularization strength λ.

2. The best setting from 40 rounds of Bayesian optimization, making use of the reusable holdout.

For both of these models, we used five-fold cross validation on the training/validation data to estimate the generalization performance of each model. Depending on the features used, the number of parameters in any one LR setting might be between 10,000 and 85,000.

3.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are functions from sequences of vectors to sequences of vectors. Because our instances are more naturally understood as sequences of symbols, the first step in applying RNNs to our problem is to map each word to a vector representation. There is a cottage industry of approaches for doing this, many of which make use of large, unannotated corpora to "embed" words as vectors. Within our Bayesian optimization setup, we consider a few such mappings.

Let $\mathbf{x}_j$ be the vector representation of the jth word token in a given response. (Let J be the length of that response, in word tokens.) To make a prediction, we use the following recurrence to define a sequence of "hidden" vectors, which can be thought of as encoding a representation of all

inputs seen sequentially up to that point:

$$\mathbf{h}_j = \tanh\left(\mathbf{W_x}\mathbf{x}_j + \mathbf{W_h}\mathbf{h}_{j-1} + \mathbf{b_h}\right) \tag{5}$$

These vectors are combined to give the "output" vector $\mathbf{o}$:

$$\mathbf{o} = \text{sigmoid}\left(\sum_{j=1}^{J} \mathbf{a}_j \odot \left(\mathbf{W_o}\mathbf{h}_j + \mathbf{b_o}\right)\right) \tag{6}$$

(In the above, the tanh and sigmoid functions are element-wise, and $\odot$ is the element-wise (Hadamard) product.)[3] The parameters of the model that are trained include all $\mathbf{W}_*$ matrices and $\mathbf{b}_*$ vectors. The $\mathbf{a}_*$ vectors might select the last element $j = J$ and ignore the rest, or take an element-wise maximum, or might be defined as another collection of parameters—this is one of the choices left to the Bayesian optimizer, along with the dimensionality of $\mathbf{h}_*$.

The output vector $\mathbf{o}$ will contain values in $[0, 1]$; we threshold element-wise at 0.5 to decide which labels are predicted:

$$\hat{\mathbf{z}} = \text{round}(\mathbf{o}) \tag{7}$$

Parameter estimation is carried out by stochastic gradient descent on the logistic loss (similar to LR) with early stopping when performance does not improve on the validation set for a certain number of steps (to prevent overfitting). In any setting, the training iterate with the best validation performance is the one that is used. Because of the large amount of time required to train a single RNN model, only a single fold of the training data is used as a validation set, with the other four-fifths used as training data (as opposed to the 5-fold cross validation used for logistic regression).

We consider many variations on the RNN described above, including the long short-term memory (LSTM) unit (a more complicated, heavily parameterized way of combining $\mathbf{x}_j$ and $\mathbf{h}_{j-1}$); for additional details see Appendix D. Our Bayesian optimizer handles a wide range of decisions about the RNN architecture; see Table 3. The distribution (Dist.) column refers to the type of prior distribution placed on the hyperparameter in the search space. More details on these hyperparameters can be found in Appendix E.

Although the size of this configuration space could be made artificially commensurate with the search space for logistic regression feature selection presented above, the key difference is in the interactions between settings. Adding or dropping a single feature will generally have a small impact on the performance of a linear model. The settings for RNNs, by contrast, are highly interdependent. For the LSTM, in particular, the appropriate learning rate depends on the size of the hidden nodes, and many other settings, and even a small change could lead to a completely ineffective model.

Virtually all RNN settings in this search space, however, possess many more parameters to be learned than the linear models. The most complicated RNN in our search space would have 3,445,475 parameters. This is 40 times more than the largest linear model, and more than 300 times larger than the unigram model. Given the limited amount of data we have, we are quite skeptical that good generalization performance can be achieved in such a setting. In our experiments, we report two types of RNNs: the "basic RNN", as described by Equations 5–7, and the equivalent network using LSTM connections (as described in Appendix D).

[3] This is slightly different from the form that is more commonly seen, which applies the linear transformation after combining the hidden nodes (i.e. $o = \text{sigmoid}(\mathbf{W_o}(\sum_{j=1}^{J} \mathbf{a_j} \odot \mathbf{h_j}) + \mathbf{b_o})$). The model presented here, however, has the advantage of being more interpretable, in that the values of $\mathbf{W_o}\mathbf{h}_j + \mathbf{b_o}$ can be directly inspected to determine the contribution that each hidden node is making to the final prediction. We make use of this to help interpret the output of these models in section 4.3 below.

Hyperparameter	Dist.	Range or Choices
Minimum word threshold	MN	{1, 2, 3, 4}
Word vectors	MN	{Google News, ANES, Reddit, ANES+Reddit}
Extra OOV dimension	MN	{Y, N}
Use Xavier initialization	MN	{Y, N}
if not using Xavier init.: Initialization scale	U	[0, 1]
Input window size	MN	{1, 3}
Add dataset type	MN	{Y, N}
Pooling method	MN	{Max, Last, Learned}
Bidirectional	MN	{Y, N}
if Bidirectional: Combine layers	MN	{Max, Mean, Concatenate}
Hidden node dimension (Basic)	qU	{5, 10, 15, . . . , 200}
Hidden node dimension (LSTM)	qU	{5, 10, 15, . . . , 100}
Train embeddings	MN	{Y, N}
Learning rate (Basic)	LU	$[e^{-4}, e^{-1}]$
Learning rate (LSTM)	LU	$[e^{-5}, e^{-1.5}]$
if Train embeddings: LR embedding factor	U	[0, 1]
LR decay delay	MN	{3, 4, 5, 6, 7, 8, 9}
LR decay factor	U	[0, 1]
Random OOV noise	MN	{Y, N}
if Random OOV noise: Noise prob	LU	$[e^{-6}, e^{-3}]$
Minibatch size	MN	{1, 4, 16}
Clip gradients	MN	{Y, N}

Table 3: Configuration space for RNNs. "Dist." refers to the prior distribution placed on the hyperparameter in the search space: U = uniform, LU = log-uniform, MN = multinomial, qU = quantized uniform.

4 Experiments

A summary of the main results is shown in Table 4. As can be seen, the more-highly featurized linear models developed using Bayesian optimization outperform the basic unigram models trained using cross validation on all but one dataset. The basic RNN models, by contrast, are in most cases considerably worse than the latter, and sometimes worse than the basic unigram model. The LSTM models show highly varied performance, sometimes winning out and sometimes performing quite poorly.

4.1 Which settings are selected?

Readers experienced with text classification will be unsurprised that no consistent pattern emerges in the best settings found by Bayesian optimization across tasks. Regardless of the type of model one chooses, the best setting of hyperparameter-like choices is a matter of empirical testing, not problem-specific or linguistic insight.

For linear models, Table 5 shows that, in addition to unigrams, commonly selected features include Brown clusters (7 out of 10 datasets), named entities (6 out of 10), part of speech tags (5 out of 10), and bigrams (4 out of 10). Brown clusters in particular are interesting, as they abstract across similar words, including misspelled variants.

The equivalent table of selected features for the best performing RNNs is shown in Table 6. Although there is no hyperparameter which is consistently selected in all cases, there are some dominant patterns which emerge. Settings which were selected in the best model for at least 7 out of 10 datasest include LSTM networks (as opposed to basic RNNs), including the word embeddings

Dataset	General Election	Primary Election	Party (Dis)likes	Person (Dis)likes	Terrorists
# Responses	239	288	4407	4684	2100
# Labels	41	42	33	34	28
Label cardinality	2.89	2.22	2.29	2.39	1.96
Unigram model	0.485	0.601	0.648	0.689	0.806
Best linear model	**0.547**	**0.667**	0.675	**0.714**	**0.811**
Best basic RNN	0.415	0.614	0.642	0.681	0.748
Best LSTM	0.313	0.433	**0.712**	0.668	0.759

Dataset	Important Issues	Knowledge: Brown	Knowledge: Cheney	Knowledge: Pelosi	Knowledge: Roberts
# Responses	8399	2098	2095	2096	2094
# Labels	74	14	12	15	14
Label cardinality	1.33	1.37	1.20	1.37	1.34
Unigram model	**0.860**	0.930	0.947	0.888	0.927
Best linear model	0.855	**0.943**	0.956	**0.930**	**0.959**
Best basic RNN	0.823	0.921	0.954	0.892	0.930
Best LSTM	0.826	0.922	**0.963**	0.896	0.946

Table 4: Results (mean F_1) for the best of various methods on each group of questions, with the best result for each dataset shown in bold. Label cardinality refers to the average number of positive labels per response.

Dataset	General Election	Primary Election	Party (Dis)likes	Person (Dis)likes	Terrorists
Unigrams	✓(B4)	✓(2)	✓(1)	✓(1)	✓(1)
Bigrams			✓(B5)		
Conjoined unigrams			✓(B2)	✓(2)	n/a
Brown clusters		✓(B)		✓(B)	
Part-of-speech tags			✓		✓
Named entities	✓		✓	✓	
Question name			✓	✓	n/a

Dataset	Important Issues	Knowledge: Brown	Knowledge: Cheney	Knowledge: Pelosi	Knowledge: Roberts
Unigrams	✓(1)	✓(1)	✓(B1)	✓(3)	✓(4)
Bigrams		✓(B5)	✓(B3)		✓(B3)
Conjoined unigrams		n/a	n/a	n/a	n/a
Brown clusters	✓	✓	✓	✓(B)	✓
Part-of-speech tags		✓		✓(B)	✓
Named entities		✓	✓	✓	
Question name		n/a	n/a	n/a	n/a

Table 5: Features selected through Bayesian optimization for logistic regression models. The number in parentheses is the threshold for number of occurrences in the corpus below which rare words were excluded. "B" indicates that the counts were binarized.

as parameters in the model, combining hidden nodes through an element-wise maximum, not using Xavier initialization, replacing random words in the input data with an out-of-vocabulary symbol, using a minibatch size of 1, and clipping the gradients. 7 out of 10 of the best RNN models were also bidirectional, with concatenation being the most common way of combining layers, though both the max and mean operations were also selected. Google's pre-trained word vectors were the most commonly selected word vectors, but the custom vectors trained on either the ANES or the Reddit datasets were also selected.

It is encouraging that we see some consistent patterns, but we are hesitant to draw any conclusions from this. Restricting the search space to some of the more popular options might improve the results, but for most datasets, there are other, very different networks discovered through Bayesian optimization that offer nearly equivalent performance. Without additional experimentation, it is very difficult to give general advice. Our recommendation, instead, is to consider using Bayesian optimization or similar hyperparameter search methods, and budget as much time as possible for a search through settings.

4.2 Should we abandon RNNs for this task?

Impressive results have recently been reported on a wide range of NLP (and other) tasks, making use of RNNs (Bahdanau et al. (2014), Vinyals et al. (2015), Ballesteros et al. (2015)). The open-ended survey response coding task differs from those success stories in two important ways: complexity of the task and the hyperparameter search process. First, the amount of training data available is relatively small (a few thousand instances), and the multi-label output scheme makes learning even more challenging. Of course, linear models profit from more training data as well. A reasonable predictor of performance is the amount of training data relative to the number of labels. The best predictor, however, is the label cardinality – the average number of positive labels per response.

With respect to the first measure, the two Likes / Dislikes datasets are considerable outliers. We would expect all classifiers to perform better on these datasets given the large amount of data and relatively small amount of labels. One possible explanation for this is that the four questions being combined for each for of these datasets contain words that depend on the question being asked. For example, "Republicans" could either refer to the subject of the question, or the opposition party, depending on the question. This is further suggested by the fact that these are the only two datasets to make use of unigrams conjoined with question type as features.

Although the above is somewhat speculative, it also provides a possible explanation for why an LSTM was able to outperform the best logistic regression model by a reasonable margin on one of these datasets. Because neural networks are able to potentially make use of arbitrary conjunctions of features, it is possible that the LSTM is doing a better job of capturing the relevance of particular words in context. Indeed there are instances for this model where the word "Republican" (in context) correctly activates the *Party* label (the party being asked about) or the *Another party* label, depending on the question.

On the other hand, our experiments' forty iterations of Bayesian optimization are almost surely not enough to find a competitive setting, given the huge space of possible settings that can be considered.[4] If we look at the progress of Bayesian optimization on logistic regression (Figure 1), we see that most of the improvements come early on (in the first 20 iterations). The few gains after that tend to be quite small. By contrast, the same plot for RNNs shows both more frequent and larger gains after the twentieth iteration. This suggests that more iterations of Bayesian optimization

[4]Bergstra et al. (2011), for example, used 80 iterations of Bayesian optimization. Many papers provide relatively few details about how hyperparameters were chosen; Weiss et al. (2015), however, demonstrate that there can be large benefits to devoting extensive (industrial-scale) computational resources to this purpose.

Dataset	General Election	Primary Election	Party (Dis)likes	Person (Dis)likes	Terrorists
RNN type	Basic	Basic	LSTM	Basic	LSTM
Minimum word threshold	2	1	4	2	1
Word vectors	Google	ANES	Google	Google	ANES
Extra OOV dimension	Yes	No	No	No	Yes
Use Xavier initialization	No	No	Yes	No	No
Initialization scale	0.83	0.77	n/a	0.3	0.09
Input window size	1	1	1	3	3
Add dataset type	No	No	Yes	Yes	n/a
Pooling method	Max	Max	Max	Max	Max
Bidirectional	No	Yes	Yes	No	Yes
Combine layers	n/a	Max	Concat	n/a	Concat
Hidden node dimension	65	170	35	135	65
Train embeddings	Yes	Yes	No	Yes	Yes
Learning rate	0.21	0.17	0.01	0.23	0.17
LR embedding factor	0.59	0.81	n/a	0.8	0.63
LR decay delay	8	6	6	5	9
LR decay factor	0.43	0.52	0.49	0.44	0.36
Random OOV noise	No	Yes	Yes	No	Yes
OOV noise prob	n/a	0.048	0.003	n/a	0.005
Minibatch size	1	1	1	1	1
Clip gradients	Yes	Yes	No	Yes	Yes

Dataset	Important Issues	Knowledge: Brown	Knowledge: Cheney	Knowledge: Pelosi	Knowledge: Roberts
RNN type	LSTM	LSTM	LSTM	LSTM	LSTM
Minimum word threshold	1	3	2	3	2
Word vectors	Google	Google	Reddit	Google	Reddit
Extra OOV dimension	No	No	No	Yes	Yes
Use Xavier initialization	No	Yes	Yes	No	No
Initialization scale	0.01	n/a	n/a	0.45	0.10
Input window size	1	3	3	1	3
Add dataset type	No	n/a	n/a	n/a	n/a
Pooling method	Learned	Max	Max	Last	Max
Bidirectional	No	Yes	Yes	Yes	Yes
Combine layers	n/a	Concat	Concat	Max	Mean
Hidden node dimension	75	70	55	45	75
Train embeddings	Yes	No	Yes	Yes	Yes
Learning rate	0.06	0.14	0.01	0.08	0.09
LR embedding factor	0.79	n/a	0.58	0.84	0.97
LR decay delay	5	9	5	6	8
LR decay factor	0.62	0.98	0.51	0.34	0.65
Random OOV noise	Yes	Yes	Yes	Yes	Yes
OOV noise prob	0.009	0.018	0.030	0.017	0.036
Minibatch size	1	1	4	4	4
Clip gradients	No	Yes	No	Yes	Yes

Table 6: Selected features for the best RNNs.

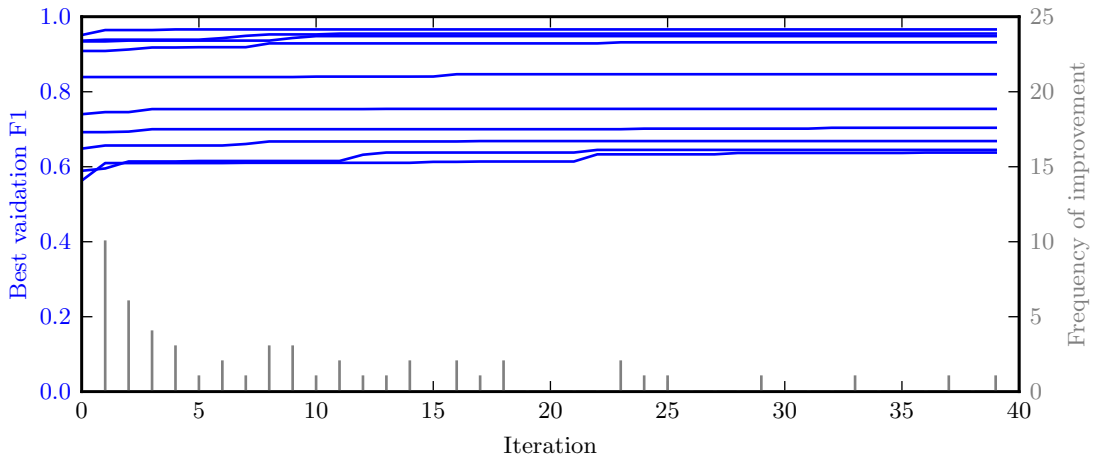

Figure 1: Progress of logistic regression on the 10 datasets across 40 iterations of Bayesian optimization. Each blue line corresponds to one dataset, and shows the current best validation performance at each iteration. The vertical grey lines show the number of datasets which improved at that iteration.

would be beneficial. Additional "tricks" that help the RNNs to converge to a good solution for a given set of hyperparameters also have the potential to improve overall performance.

Neural networks are often billed as a way to avoid manual feature engineering. We believe our experiments provide a useful illustration that this claim is misleading. Both linear and neural models require many choices to be made; we have tried to carry out a fair comparison by using the best-available techniques to *automate* those choices. Explicitly laying out the space of choices to be made for each reveals a much larger space of options for RNNs. This is not to say that the search for a good setting is hopeless, merely that our experiments did not reveal settings competitive with the best discoverable linear models, in most cases.

There are still more options for training neural networks that our Bayesian optimizer did not consider, including the use of regularization, dropout, additional layers, different learning rate schedules, and so on. Because some runs of our learning procedure fail to make any progress (possibly due to a poor random initialization), the search procedure learns very little from these iterations. It is possible that better learning regimens, with (for example) more time per iteration of Bayesian optimization, and, as noted above, increasing the number of iterations of Bayesian optimization would allow us to discover more competitive RNN settings for this dataset.

Of course, there is no reason we necessarily have to choose one over the other. It is entirely possible to include the choice between logistic regression and neural networks (or other classifiers) as yet another part of the configuration space for Bayesian optimization, and thereby automate this choice as well. When using RNNs, however, it seems as though the time required for training will tend to be the limiting factor in the search for the best model for the data.

At this juncture, we do not recommend abandoning RNNs for multi-label text classification problems in general, or the open-ended survey response coding tasks considered here in particular. We do, however, caution researchers that the daunting space of hyperparameter choices may be prohibitive to search, even with automated techniques, and linear models remain a good default from the computational point of view. At the very least, given how easy and cheap it is to train linear models, they should always be used as a baseline for the purpose of comparison, and an automated model selection process helps to ensure that this is a reasonable comparison.

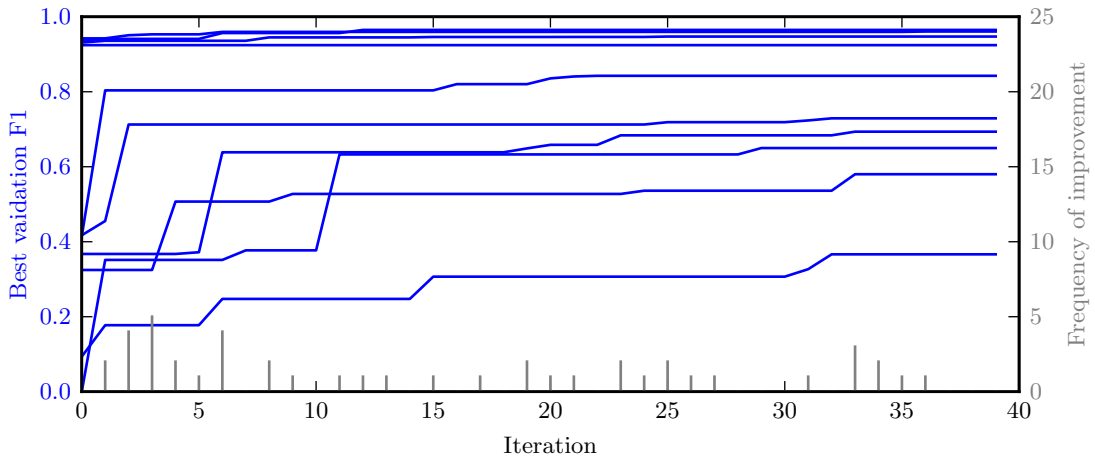

Figure 2: Progress of LSTMs on the various datasets across 40 iterations of Bayesian optimization. Each blue line corresponds to one run of a LSTM on a single dataset, and shows the current best validation performance at each iteration. The vertical grey lines show the number of datasets which improved at that iteration.

4.3 What about interpretability?

One of the difficulties of working with neural networks, or other non-linear, highly parametrized models (such as random forests), is that they are generally thought to be difficult to interpret. Although linear models such as logistic regression might make predictions that seem highly counterintuitive, it is at least easy to determine precisely why a classification was made, simply by identifying the weights associated with the various features that are present. With RNNs, on the other hand, there might be thousands of interacting parameters that are relevant for a single example.

As a way of exploring this issue further, we present one in-depth example from this classification task, to better understand the differences between the two approaches.

Consider the following response to the question *Is there anything in particular that you don't like about the Republican party?*: "1disaprove of the stance on gay rights issues, energy policy, religion permiates the rep. party." This response is typical, in that it contains typos, touches on multiple issues, and is not entirely grammatical.

Human coders assigned six codes to this response: Party [the Republican Party], Religion, Policy-Liberty, Policy-Other, Groups, and Other. Both the best LSTM and the unigram model successfully predicted all of these but "Other", and predicted no false positives. (The best logistic regression model found through Bayesian optimization is very similar to the unigram model in this case, but the latter will be used for simplicity of presentation).

Figure 3 presents an illustration of how this example is processed by the linear model; colors correspond to $\beta_0 + \boldsymbol{\beta}^T \mathbf{x_i}$ in equation 2, with strongly positive words in bright blue and strongly negative words in bright red. As can be seen, for at least four of the labels, there is a single word which dominates the decision, with obvious relevance to the label.

Figure 4 presents the equivalent picture for the best performing LSTM on this dataset. Because this network uses a max operation over hidden nodes, there are no "negative" elements, and we have instead plotted the probability of positive classification for each element in the sequence, from white (0) to bright blue (1). These values correspond to the output of equation 6 if we don't perform the weighted sum over elements (i.e. the values for word j are given by $\mathbf{o_j} = \text{sigmoid}(\mathbf{W_o h_j} + \mathbf{b_o})$).

Also, because this is a bidirectional network with concatenated layers, the value for each element

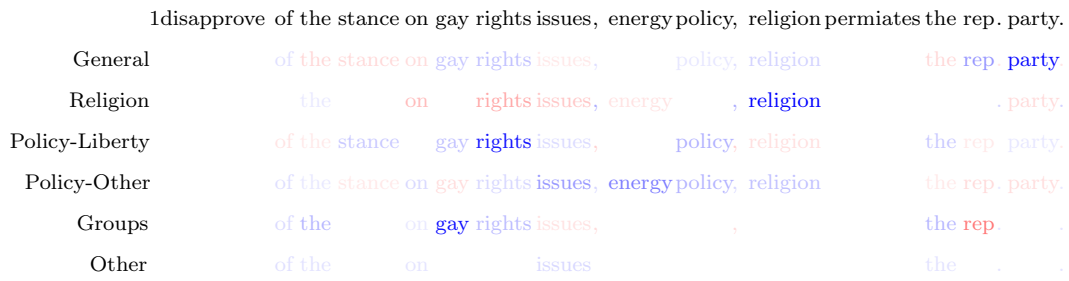

Figure 3: Feature (word) weights for the unigram models (for a subset of labels) applied to the phrase "1disaprove of the stance on gay rights issues, energy policy, religion permiates the rep. party." In this visualization, blue indicates a positive coefficient, and red indicates a negative one, with bolder colors indicating greater magnitude.

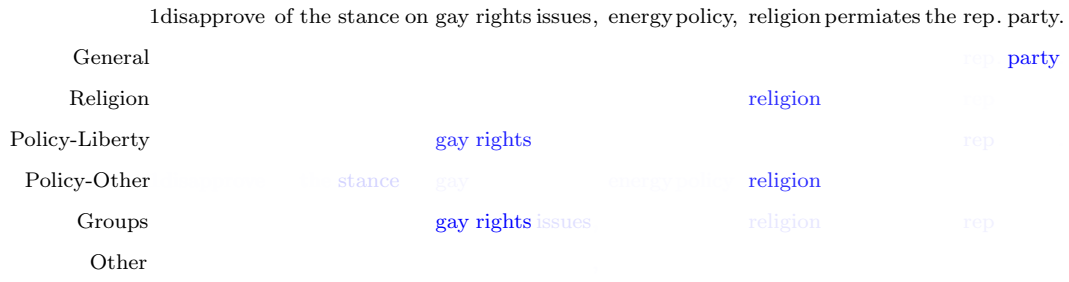

Figure 4: Output probabilities (for a subset of labels) associated with each element in the sequence from the best performing RNN (a bidirectional LSTM), applied to the phrase "1disaprove of the stance on gay rights issues, energy policy, religion permiates the rep. party." White corresponds to 0, bright blue to 1.

represents not just the corresponding word, but the word in its full context, from the start of the sentence in one direction, and from the end of the sentence in the other. As a result, for example, we see that both words of the bigram "gay rights" are prominent for both the Liberty and Groups labels.

Although most of the positive probabilities here again make sense for each label, the high probability of the "Policy-Other" label for the word "Religion" seems strange. Moreover, it is unclear from all of this whether the contribution of each element depends on its context or not. For example, would "party" cue the General label so strongly if not preceded by "rep."? Would "gay" cue Liberty so strongly if not followed by "rights"?

It is possible to investigate these questions by running an altered versions of the sentence through the final model. Figure 5 shows the result if remove the words "rights" and "rep.", leaving the semantically similar sentence "1disaprove of the stance on gay issues, energy policy, religion permiates the party."

As can be seen, several things have changed. The importance of "party" has dropped off dramatically without the context of "rep.", such that the General label would no longer be predicted.

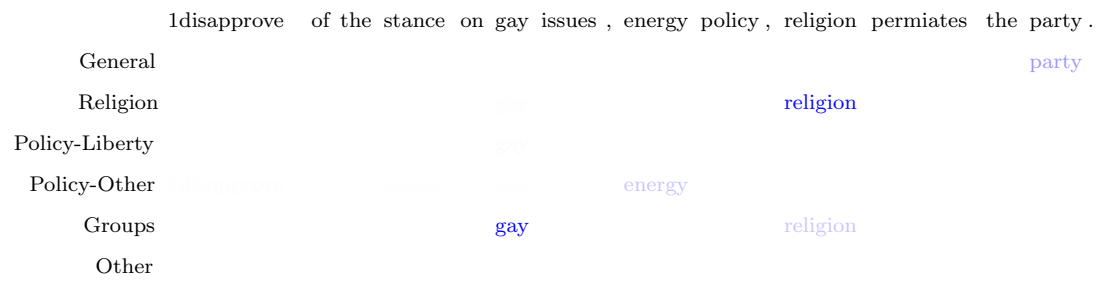

Figure 5: Modified output probabilities (for a subset of labels) associated with each element in the sequence from the best performing RNN (a bidirectional LSTM), applied to the modified phrase "1disaprove of the stance on gay issues, energy policy, religion permiates the party." White corresponds to 0, bright blue to 1.

Religion and Groups are unaffected, but Policy-Liberty drops effectively to zero, as we would expect, without the word "rights". Most surprisingly, the prediction of Policy-Other associated with the word "religion" has also disappeared, even though that word was three words away from any edits.

This example illustrates that we can to some extent make sense of what is happening with the internals of RNNs, but that it is difficult to disambiguate the contribution of each element of the sequence or to predict how the prediction might differ for a slightly different sentence.

4.4 Is the Reusable Holdout working?

As described in (Dwork et al., 2015), the Reusable Holdout (RH) is a technique designed to limit the probability of overfitting to a validation set, when repeatedly evaluating models in an adaptive fashion. This makes the technique ideally suited for use with Bayesian optimization, where the next set of hyperparameter values to try is chosen based on all evaluations of the validation set seen so far. Full details are provided in Appendix C.

If the RH is working, we would expect that the final test-set performance of the best model from a model search using the RH would be better than that of the best model found without using it. Although there was insufficient time to replicate all experiments with and without the RH, we did replicate the search for the best logistic regression model.

The results of this comparison are shown in Table 7. For 8 out of 10 datasets, making use of the RH resulted in a model with a final test performance that was equal to or greater than the best model produced without using the RH, though in most cases the differences are negligible (< 0.01 mean F_1). Thus, while it is encouraging that using the RH tends to give slightly better results most of the time, we have no conclusive evidence that it is actually helping to prevent overfitting. Note that this is not intended to be a rigorous evaluation of the RH technique, but a secondary observation on a method we have chosen to use.

One possible explanation for this result is that overfitting is not a serious problem for logistic regression models with these datasets. On the other hand, it is also possible that because we are making more evaluations of the validation set that would strictly be allowed by the theory (as do Dwork et al.), we cannot depend on the guarantees of the RH. In other words, further work is required to further connect the theoretical and experimental properties of the RH, and additional experiments are needed to determine empirically how far we can push it beyond its theoretical limitations.

Dataset	General Election	Primary Election	Party (Dis)likes	Person (Dis)likes	Terrorists
With RH	**0.547**	**0.667**	**0.675**	0.714	**0.811**
Without RH	0.494	0.639	0.670	**0.742**	0.785
Dataset	Important Issues	Knowledge: Brown	Knowledge: Cheney	Knowledge: Pelosi	Knowledge: Roberts
With RH	**0.855**	**0.943**	**0.956**	0.930	**0.959**
Without RH	0.855	0.941	0.955	**0.935**	0.955

Table 7: Comparison of results (mean F_1) for the best logistic regression model found through Bayesian optimization with and without using the Reusable Holdout. The better result for each dataset is shown in bold.

5 Limitations

There were a number of limitations to this project. The first set of limitations have to do with the data. Although the ANES conducts surveys during every national election, we only had access to the responses from a single survey, rather than having data from multiple years. In addition, as described above, we only had access to the interviewer summaries, written during the interviews, rather than the full transcripts, which might have been much richer. The errors in these summaries (spelling and grammar) caused some difficulty, and greater human preprocessing might have produced improvements. Furthermore, for the annotations, we only had access of the final agreed-upon annotations from the human coders, rather than the specifics of cases where coders initially disagreed, which might have been useful in trying to understanding cases where the coding is ambiguous.

Second, in terms of scope, we only considered two classes of algorithms to apply to this dataset. Although there are many other options we could have considered, these two served effectively as a strong baseline, on the one hand, and a chance to explore more recently-developed but promising methods, on the other. If we were only interested in overall performance, it would make sense to consider a wider range of options, all of which could be considered within the framework of Bayesian optimization. Similarly, additional configurations of RNNs could have been considered, such as using additional layers, or other architectures, such as convolutional neural networks.

Third, in terms of performance, we were limited to some extent by computational considerations. It is likely that the RNNs in particular would have benefited from additional computational power (for additional runs of Bayesian optimization), as well as additional refinement of the methods used for training. Because training time for these models can be quite long, there were only a limited number of iterations that could be tried on all datasets. Further experience with these methods will likely produce faster training, but to scale up to much larger datasets would likely require further investment in processing power.

6 Conclusions

Multi-label text classification remains a fundamental paradigm into which a variety of computational social science problems can be cast. We have considered the use of two very different approaches to the automated coding of a modest corpus of open-ended survey responses.

As expected, we find that traditional NLP methods offer reasonable performance on this task, particularly in the case of large amounts of training data and low label cardinality. Although reasonable baseline performance can be obtained with a simple unigram model, it is generally possible to realize gains through the use of additional features, such as bigrams and Brown clusters.

Though recurrent neural networks have recently demonstrated impressive gains on a variety of NLP tasks, we have largely been unable to demonstrate the same sorts of benefits on this dataset, potentially as a result of the limited amount of data available for training these complex and highly-parameterized models.

Nevertheless, the potential for gains is still there, as demonstrated by a sizable performance improvement on one of the ten datasets examined in this report. We remain optimistic that with additional refinement, these methods may be widely applicable to a broad range of problems. For the moment, however, difficulties related to computational demands, interpretability, and the large number of possible configurations suggest that they are perhaps not yet ready for convenient, off-the-shelf use in computational social science applications.

Finally, we have fruitfully demonstrated the application of two relatively simple techniques, which can be combined with any existing machine learning methods for a variety of problems. In particular, Bayesian optimization provides a general framework to automatically determine the settings to use for a particular method for good generalization performance. Used in tandem with Bayesian optimization, the reusable holdout appears to offer some protection against overfitting to the validation set, to help ensure that the resulting models will truly generalize.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1), 2015.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NIPS)*. 2011.

Peter F. Brown, Peter V. de Souza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based N-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013. arXiv:1301.3781 [cs.CL].

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*. 2012.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Sida Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2012.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

A Survey questions

- Dataset 1: General Election
  - Why do you think Barack Obama won the Presidential election?
  - Why do you think John McCain lost the Presidential election?
- Dataset 2: Primary Election
  - Why do you think Barack Obama won the Democratic nomination?
  - Why do you think Hillary Clinton lost the Democratic nomination?
- Dataset 3: Party (Dis)likes:
  - Is there anything in particular that you like about the Democratic party?
  - Is there anything in particular that you like about the Republican party?
  - Is there anything in particular that you don't like about the Democratic party?
  - Is there anything in particular that you don't like about the Republican party?
- Dataset 4: Person (Dis)likes:
  - Is there anything in particular about John McCain that might make you want to vote for him?
  - Is there anything in particular about Barack Obama that might make you want to vote for him?
  - Is there anything in particular about John McCain that might make you want to vote against him?
  - Is there anything in particular about Barack Obama that might make you want to vote against him?
- Dataset 5: Terrorists
  - As you know, on September 11th 2001, a group of Terrorists took control of several U.S. commercial airplanes and crashed them into the World Trade Center in New York and the Pentagon in Washington. What do you think the Terrorists were trying to accomplish by their actions?

- Dataset 6: Important Issues
  - What has been the most important issue to you personally in this election?
  - What has been the second most important issue to you personally in this election?
  - What do you think is the most important political problem facing the United States today?
  - What do you think is the second most important political problem facing the United States today?
- Dataset 7: Political Knowledge: Brown
  - What job or political office does Gordon Brown now hold?
- Dataset 8: Political Knowledge: Cheney
  - What job or political office does Dick Cheney now hold?
- Dataset 9: Political Knowledge: Pelosi
  - What job or political office does Nancy Pelosi now hold?
- Dataset 10: Political Knowledge: Roberts
  - What job or political office does John Roberts now hold?

B Bayesian optimization for hyperparameter selection

A critical part of model specification is the selection of hyperparameter settings, such as regularization strength, or the number of layers in a neural network. For complex models, this can involve choosing values for potentially dozens of inter-dependent hyperparameters. Moreover, training and evaluating a model for a particular choice of hyperparameters is typically a costly operation. Thus, only a limited number of choices can realistically be evaluated.

The three main techniques that are typically used for this task are human-guided search, grid search, and random search. In at least one study (Bergstra and Bengio, 2012), random search was found to outperform grid search. More recently, a collection of approaches known using Bayesian optimization have emerged as a strong alternative. Because the true mapping of hyperparameter settings to performance is complex and unknown, Bayesian optimization replaces this function with a more manageable surrogate function, which it then tries to optimize.

If we define $\mathcal{X}$ as the full space of possible hyperparameters to be searched, and $y = f(x)$ as the model performance for a particular set of hyperparmater values, x, then the *expected improvement* criterion for Bayesian optimization tries to choose new set of value x that will will return the highest expected increase over the current best result, y^*. More precisely, for the case where we want to maximize y, the expected improvement is given by:

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{\infty} \max(y - y^*, 0) p(y|x) dy$$

The Tree-structured Parzen Estimator (TPE) is an approach to Bayesian optimization which attempts to maximize expected improvement by modeling $p(x|y)$ and $p(y)$ (Bergstra et al., 2011). In particular, given a set of results $\mathcal{Y}$ returned so far, TPE take y^* not as the best, but as some

quantile of these results. $p(x|y)$ is then modeled using two non-parametric density estimators, one for $y < y^*$ and one for $y \geq y^*$.

The space of $\mathcal{X}$ is constructed as a tree (because some hyperparamters depend on particular settings of others). At each node of the tree, a prior distribution is placed over the possible values of the corresponding hyperparamter, using one of four probability distributions: uniform (U), log-uniform (LU), quantized uniform (qU) or a multinomial distribution (MN) for discrete-valued hyperparameters. Hyperparameter values can be drawn from the tree according to these distributions, which will be updated throughout the course of optimization. The likelihood is modeled as a mixture of Gaussians, with one placed at each point explored in the search space. Choosing a new point to explore in the space is done by randomly drawing many points and choosing the one with the highest expected improvement, which can be reduced to the ratio between the values of the two non-parametric densities at that point.

Berstra et al (2011) used this methodology to train two Deep Belief Networks. They generated 30 random trials and then ran 50 trials of Bayesian optimization. Using this method, they found that the TPE approach outperformed random search, human search, and Gaussian processes, discovering a new state-of-the-art result on one dataset. Even for a large search space, the computation required to choose the next set of hyperparameter values to try is extremely fast, thus evaluating the model for those settings will generally be the bottle-neck. Importantly, however, this search process only considers each hyperparameter in isolation, and does not model interactions between hyperparameters, except in the case of dependence between parent and child nodes.

For this analysis, we make use of a python package called `hyperopt`, which carries out the iterative selection of hyperparameter values using the TPE search on a user-defined search space (Bergstra et al., 2015).

C Reusable Holdout

Because Bayesian optimization works by repeatedly constructing a classifier and evaluating it on a validation set, there is an inherent danger of overfitting to this validation set, just as individual classifiers tend to overfit to the training data. As a counter-weight to this, we make use of a recently-proposed idea called the reusable holdout (Dwork et al., 2015). Based on an idea from differential privacy, this technique tries to limit the amount that can be learned about the individual items in the validation set, and thereby prevent overfitting.

The algorithm is as follows: on each iteration, a noisy threshold is generated, centered on some positive value. Both the training error and the validation error are then computed, but the later is not returned directly. Rather, if the difference between the training error and the validation error is below the current threshold, the training error is returned instead. If it is larger than the threshold, then some noise is added to the validation error, and that quantity is returned, and the threshold is reset to the original value plus some noise. Dwork et al. demonstrate that they can bound the probability of overfitting for a number of evaluations of the validation set which is exponential in the size of the validation set, even though decisions about the model are being made adaptively.

For this analysis, we apply the reusable holdout using the suggested default parameters to the values returned from each iteration of Bayesian optimization. These noisy values are then used by the Bayesian optimizer to choose the next set of hyperparameter values to evaluate. To choose a final model, we use the true validation values (without noise), and select the model with the best validation performance. This model is then evaluated on the test data, without any additional noise.

Although one can calculate a "budget", the precise number of evaluation of the validation set which are allowed while controlling the probability of overfitting, Dwork et al. ignore this in their

experiments, and demonstrate that this technique works empirically, even when using an excess number of evaluations. We do the same here, ignoring the budget, and apply the RH on all rounds of Bayesian optimization.

Reusable Holdout algorithm (ignoring budget restrictions)

1. Set base threshold T, and noise level σ.
2. Sample $\gamma \sim \text{Laplace}(2 \cdot \sigma)$
3. $\hat{T} \leftarrow T + \gamma$
4. For each query:
   (a) Compute training and validation performance ($\mathcal{E}_t$ and $\mathcal{E}_v$)
   (b) Sample $\eta \sim \text{Laplace}(4 \cdot \sigma)$
   (c) If $|\mathcal{E}_t - \mathcal{E}_v| > \hat{T} + \eta$
      i. Sample $\xi \sim \text{Laplace}(\sigma)$ and $\gamma \sim \text{Laplace}(2 \cdot \sigma)$
      ii. $\hat{T} \leftarrow T + \gamma$
      iii. Output $\mathcal{E}_v + \xi$
   (d) Else, output $\mathcal{E}_t$

D Long Short-Term Memory (LSTM) networks

One problem with basic RNNs is the so-called "vanishing gradient" problem, in which the gradients for parameters near the beginning of the recurrent network end up being vanishingly small. A more complicated model designed to overcome this difficulty is known as the LSTM (Graves, 2012; Hochreiter and Schmidhuber, 1997). This model involves a drop-in replacement for the way in which successive hidden nodes are produced from a previous hidden node and the next sequence element. In particular, the LSTM introduces a "memory node" in parallel to each hidden node, as well as a series of "gated connections", which can be conceptualized as controlling the flow of information among the previous and current hidden nodes, the previous and current memory nodes, and the next sequence element. These gates are vectors of the same size as the hidden nodes, and take values in the range $[0, 1]^d$.

As with the other components in an RNN, we introduce additional matrices and vectors to encode linear transformations. Let $\mathbf{c_j}$ be the memory node for the jth word in a sequence, let $\mathbf{i}$, $\mathbf{f}$, and $\mathbf{o}$, be the input, forget, and output gates, respectively, and let $\mathbf{g}$ be an intermediate vector (used to simplify the presentation). The computation of $\mathbf{g}$ is identical to the way in which the next hidden node is usually computed in a basic RNN. The next memory node is then computed as a (element-wise) weighted combination of $\mathbf{g}$ and the previous memory node, weighted by the the input and forget gates, respectively. The next hidden node is finally computed from the current memory node, scaled by the output gate. In this way, the memory node can in theory store information across multiple elements in the sequence, without having to output it to a hidden state.

The equations which govern these connections are as follows:

$$\mathbf{i} = \text{sigmoid}\left(\mathbf{W_{xi}x_j} + \mathbf{W_{hi}h_{j-1}} + \mathbf{W_{ci}c_{j-1}} + \mathbf{b_i}\right) \tag{8}$$

$$\mathbf{f} = \text{sigmoid}\left(\mathbf{W_{xf}x_j} + \mathbf{W_{hf}h_{j-1}} + \mathbf{W_{cf}c_{j-1}} + \mathbf{b_f}\right) \tag{9}$$

$$\mathbf{o} = \text{sigmoid}\left(\mathbf{W_{xo}x_j} + \mathbf{W_{ho}h_{j-1}} + \mathbf{W_{co}c_{j-1}} + \mathbf{b_o}\right) \tag{10}$$

$$\mathbf{g} = \tanh\left(\mathbf{W_{xg}x_j} + \mathbf{W_{hg}h_{j-1}} + \mathbf{b_g}\right) \tag{11}$$

$$\mathbf{c_j} = \mathbf{f} \odot \mathbf{c_{j-1}} + \mathbf{i} \odot \mathbf{g} \tag{12}$$

$$\mathbf{h_j} = \mathbf{o} \odot \mathbf{c_j} \tag{13}$$

$$\tag{14}$$

E RNN hyperparameters

Additional details about the hyperparameters used for RNNs are given below.

- Minimum word threshold: words which appear less than this number of times in the corpus are replaced with an out-of-vocabulary symbol (OOV)
- Word vectors: We consider four possible sets of word vectors, all trained with `word2vec` (Mikolov et al., 2013). One is the default vectors provided by Google, trained on Google News (approximately 100 billion words), which has by far the largest coverage. The other three were trained using the `word2vec` implementation from the `gensim` python package (Řehůřek and Sojka, 2010).[5] One of these was trained on all words in the ANES survey results (this dataset), which is very small, but captures the full vocabulary. Another was trained on all comments from the Politics subreddit on reddit.com[6], which is much smaller than the Google News corpus, but larger than the ANES data, similarly informal, and roughly in-domain. The last used the union of these two. In all cases, the dimensionality of these vectors was 300.
- Extra OOV dimension: If true, the OOV symbol is initialized as a word vector orthogonal to all others, by adding an extra dimension to the pre-trained word vectors. Otherwise, the OOV symbol is given a random vector, as are all words that don't exist in the pre-trained word vectors.
- Use Xavier initialization: Most parameters[7] of the RNN are initialized randomly by drawing from a uniform distribution in the range (-1, 1). If using Xavier initialization (Glorot and Bengio, 2010), this draw is scaled by $\sqrt{6/(n_i n_o)}$, where n_i and n_o are the number of incoming and outgoing network connections, respectively.
- Parameter initialization scale: If not using Xavier initialization, the random initializations are scaled by this hyperparameter.
- Input window size: The first layer of the network converts each element of the sequence into the corresponding word vector. If window size is equal to 3, they are instead replaced by a concatenation of that word vector with vectors for the preceding and following words.

[5] `word2vec` was used with the options `min_count=1`, `size=25`, `hs=0`, `negative=10`.

[6] `https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/`

[7] The exceptions are all bias vectors, which are initialized to zero, except for the bias term for the forget gate in the LSTM, which is randomly initialized from a uniform distribution in the range (0.8, 1.0).

- Add dataset type: If true, the characteristics of the dataset ("Likes vs. Dislikes," "Democrat vs. Republican," etc.) are represented as a binary vector and appended to the vector for each sequence element.

- Pooling method: If "last," only the last hidden node is used in computing the output. If "max," the element-wise maximum across the hidden nodes is used. If "learned," the vectors $\mathbf{a}_j$, as described in equation 6 are computed from a transformation of the corresponding hidden nodes using a linear transformation (whose parameters are also learned by the network) followed by a softmax to created a weighted combination of hidden nodes.

- Bidirectional: If true, two layers of hidden nodes are used, one computing hidden nodes forward along the sequence, and the other working backwards. The parameters for these layers are with tied (i.e. the same W_x is used for both the forward and backwards layers).

- Combine layers: For a bidirectional network, the two layers can be combined by concatenating the corresponding hidden nodes, averaging them, or taking the element-wise maximum.

- Hidden node dimension: The dimensionality of hidden nodes. The LSTM networks were restricted to smaller hidden nodes (5-100) than the basic RNN (5-200), because the LSTM involves many more parameters.

- Learning rate: This hyperparameter controls how aggressively the parameters are changed on each iteration. A value of 0.1 is often reported in the literature as being a robust choice. Parameters values for the LSTM had a tendency to diverge during learning, so the learning rate was set to be smaller than for the basic RNN.

- Train embeddings: If true, the initial word vector representations of words will be considered as parameters that can be updated during learning.

- LR embedding factor: If the word embeddings are being updated, the learning rate for the word embedding parameters is scaled by this hyperparameter (so that they are changed more slowly than the rest of the parameters).

- LR decay delay: If this many epochs pass without the performance on the validation set improving, the network parameters are reset to the previous best values, and the learning rate is reduced.

- LR decay factor: This is the amount by which the learning rate is reduced after a given number of epochs have passed with no improvement.

- Random OOV noise: If true, on each epoch, words in each input document are randomly selected to be replaced by the out-of-vocabulary symbol

- Noise prob: This is the probability with which words are randomly chosen for temporary replacement by an OOV symbol

- Minibatch size: The number of training examples to use for approximating the gradient at each iteration

- Clip gradients: If true, the gradients are truncated, element-wise, to be in the range [-1, 1].

If we include the pre-trained word embeddings in the model as parameters that can be udpated (Train embeddings = Yes), they form by far the largest set of parameters in the model. With a vocabulary of approximately 10,000 words, and 300 dimensional word vectors, the embeddings alone account for 3,000,000 parameters.

For a basic bidirectional RNN in a "typical" set up (100-dimensional hidden nodes and approximately 50 output codes, with a window size of 1), the network itself would contribute an additional 50,000 parameters. If we use an LSTM rather than a basic RNN, this adds an additional 150,000 parameters for all the internal gates. If we use a window size of 3 instead of 1, both sets of parameters increase by 120%, to approximately 110,000 parameters for the basic network, and an additional 330,000 parameters for the LSTM.