July 2021

Perspective
EXPERT INSIGHTS ON A TIMELY POLICY ISSUE

PETER SCHIRMER, AMBER JAYCOCKS, SEAN MANN, WILLIAM MARCELLINO, LUKE J. MATTHEWS, JOHN DAVID PARSONS, DAVID SCHULKER

Natural Language Processing

Security- and Defense-Related Lessons Learned

This Perspective offers a collection of lessons learned from RAND Corporation projects that employed natural language processing (NLP) tools and methods. It is written as a reference document for the practitioner and is not intended to be a primer on concepts, algorithms, or applications, nor does it purport to be a systematic inventory of all lessons relevant to NLP or data analytics. It is based on a convenience sample of NLP practitioners who spend or spent a majority of their time at RAND working on projects related to national defense, national intelligence, international security, or homeland security; thus, the lessons learned are drawn largely from projects in these areas. Although few of the lessons are applicable exclusively to the U.S. Department of Defense (DoD) and its NLP tasks, many may prove particularly salient for DoD, because its terminology is very domain-specific and full of jargon, much of its data are classified or sensitive, its computing environment is more restricted, and its information systems are generally not designed to support large-scale analysis. This Perspective addresses each of these issues and many more. The presentation prioritizes readability over literary grace.

We use *NLP* as an umbrella term for the range of tools and methods that enable computers to analyze human language. NLP tasks can be loosely binned into two categories:

1. natural language understanding (NLU), which includes document classification, information retrieval, question answering, summarization, and content analysis
2. natural language generation, which includes translation, text-to-speech, and chatbots.

NLP tasks at RAND are almost entirely of the NLU variety.

The preponderance of tasks involve some type of document classification—for example,

- classifying contract data to understand military labor force mix
- predicting military officer promotions, drawing from written assessments
- predicting the readiness of military units and organizations, drawing from commanders' comments
- identifying possible Russian trolls, drawing from tweets
- discovering conspiracy theories on social media platforms
- categorizing jobs, drawing from their descriptions in advertisements
- predicting activity by Twitter users, drawing from tweets
- determining whether psychologists used evidence-based treatment, drawing from their clinical notes
- identifying studies conducting health service research and primary care research that were supported by federal agencies.

Other NLP tasks were

- comparing language used in written evaluation reports of officers selected and not selected for battalion command
- searching and retrieving information from aircraft maintenance logs
- summarizing text and identifying topics for bioterror policy
- summarizing cloud-vendor responses to a request for information from a federal agency
- retrieving information from court cases on cyberstalking
- using news stories to answer questions about mass shootings and other major acts of violence

Abbreviations

AI	artificial intelligence
API	application programming interface
AWS	Amazon Web Services
DoD	U.S. Department of Defense
ETL	extract, transform, load
LR	logistic regression
NLP	natural language processing
NLU	natural language understanding
PII	personally identifiable information

Targeted data collection may produce better training data that in turn lead to better models by reducing the risk of diluting data with less relevant observations for the problem at hand.

- collecting documents for human review and potential inclusion in a literature review of global health programs.

Each of these projects consisted of, to a greater or lesser degree, four major steps:

1. data collection
2. data processing (i.e., NLP-specific text processing in preparation for modeling)
3. model development
4. application development and deployment.

Our descriptions of the lessons learned are organized into those four steps. We also offer some overall lessons that apply to each step or to the NLP process in general.

Data Collection

Data collection requires a strategy. In the era of big data, the prevailing attitude is that more is better. Although this might be true with unconstrained resources, as a practical matter it does not work, for several reasons. The first and most obvious reason is that resources—at least RAND's—are not unconstrained. Second, and perhaps less obvious, targeted data collection may produce better training data that in turn lead to better models by reducing the risk of diluting data with less relevant observations for the problem at hand. For a project examining conspiracy theories in social media, a Ph.D. linguist built a corpus of training data derived from a theory of human discourse that resulted in a highly accurate model for detecting conspiratorial social media posts. Critical to the effort was understanding how people talk about controversial ideas—for example, whether they are promoting these ideas, analyzing them, or countering them. This effort, in turn, required sampling posts according to stance and tone, not merely performing keyword searches for such terms as *COVID* or *deep state*.

Yet another reason that more is not always better is that the amount of time spent processing large amounts of data (the subject of the next section) can be excessive, particularly if a process cannot be performed in parallel. Text files can be quite large. Testing processing and modeling code on a small sample of data can show how large the final corpus can be before running into practical constraints on time or computer memory. One recent project examining threats from white nationalists on social media used a data set collected by a third party via an application programming interface (API). The data were annotated with stance tags—potentially helpful—but the data set was hundreds of gigabytes and took quite a long time even to download and open before the researchers could explore the data. If an organization is going to collect and curate data sets for DoD users, it would help to offer smaller representative samples of the data.

Many systems were not designed to enable data collection at scale; this is often a particular problem with systems whose data are mostly text. Unlike the private sector, DoD and other government organizations do not have a business model that derives value from their data. Many systems do not provide an API that enables automatic data pulls. One solution is to scrape websites. Some sophisticated—and often open-source—tools can do this. Selenium is a cross-platform, multilanguage (Java, Python, Ruby, etc.) website-testing tool that emulates human behavior and can also be used for web scraping. It is especially useful with dynamically generated content. We have also found that simpler scrapers, such as BeautifulSoup (a Python library), work quite well with static content. However, web scraping has significant limitations and often is not the best option. Web scraping tends to be a brittle process, and code can break if the format of the target website changes. Web scraping may even be prohibited under terms of some site licenses and subscriptions. Information on classified systems is particularly challenging to scrape because it may be impossible to install the necessary software for scraping, and server response times tend to be much slower. In some cases, we have had to resort to having a person systematically download query results, which can require anywhere from hours to days to build the data set. With systems that were not designed for data collection at scale, the best approach (but one that will not always work) is to ask the system owners to provide a copy of the data. The owners themselves could query the data using criteria you provide or simply give you a copy of the entire database if it is not too large.

Text collection and extraction may require a suite of tools, including custom algorithmic processes. Custom code may be needed to identify relevant documents from a larger collection and data from the relevant documents. Search engine results often generate a large number of

Text collection and extraction may require a suite of tools, including custom algorithmic processes. Custom code may be needed to identify relevant documents from a larger collection and data from the relevant documents.

false positives (for example, our searches for news stories about shootings inevitably retrieve stories about basketball, hockey, and golf), and it might be easier to filter through results as part of the extract, transform, load (ETL) process than to attempt to use more-precise search terms in the initial collection. Once text documents are collected, the data may be locked up in image files that are not inherently machine readable. This is particularly true for older administrative files, such as job descriptions, personnel evaluations, contracts, and work orders. If the text is not machine readable, optical character recognition (OCR) is needed. Tesseract is a very accurate cross-platform engine that can support complicated OCR tasks, such as managing layouts, tables, and graphic captions. Apache Tika is a versatile tool for extracting text from a wide variety of document types, including PDFs. Adobe Acrobat Pro can also be used to extract text from PDFs, but we found that it was preferable to use Adobe to convert PDFs to Microsoft Word documents, and then use a Python library to process the Word documents into model-ready data, because the Python library gave us more control over how we used the content in our code. Even after the OCR is performed or other initial collection and extraction are performed, extensive algorithmic procedures may be needed to extract the textual elements of interest. One of our projects required two programmers to work more than 100 person-hours to complete the data-extraction process.

Metadata or other identifiers must be collected or generated so text can be linked to other files. In addition to the content of these files, you might also need to generate metadata so you can associate the content with other data. For example, on another project, one military service shared its personnel evaluations with us, but they were stored in a separate database that was not aligned with, or linked to, the service's human resources data systems. Not only did our own team have to expend the extra effort to link the data, but the lack of linkage also demonstrated that the organization could make little use of its evaluations for talent management.

Data engineering is its own discipline, with specialized skills and tools. In many of our projects, the data engineer (the person building the platform, services, and framework to supply data for the models) and the data modeler are the same person. Data collection and ETL may take a week or so of effort, provided that data are in some machine-readable format and can be collected in bulk quantities. Data may be stored on a local laptop or in a directory shared by a research team. It may even be possible to quickly update the data, depending on how often the data change and how they are accessed. However, in some cases, this arrangement is suboptimal or even infeasible. Data may be of such a volume that they must be processed in a distributed Hadoop cluster, which requires specialized skills to even access. Data may be ephemeral, such as individual packets for officer promotion boards; data timeliness may be important, such as for monitoring news stories about terrorist attacks; and databases may simply be too large to collect in their entirety on an as-needed basis, such as with many social media platforms. In these cases, some ongoing process is needed to collect and ETL the data. This poses two challenges for us. First, it is rare for RAND researchers to work on a project in perpetuity: Instead, they typically work on several projects at once, with each project lasting anywhere from a few weeks to a few years. Second, RAND researchers are experts on particular subjects, such as military readiness or aircraft maintenance, and on methods, such as word embeddings and attention models, but less so on communicating with servers via APIs, scheduling workflows with Apache Airflow or similar tools, or managing read-write access to S3 (Simple Storage Service) buckets on Amazon Web Services (AWS). (People who are skilled in all aspects of data science are somewhat derisively referred to as *unicorns*—exceedingly rare, if they exist at all.) Some members of RAND's information services staff are dedicated to helping multiple research teams manage their data collection and storage processes, and it particularly helps if these people understand DevOps (combining software *development* and information technology *operations*) and data modeling. They can help, for example, turn a one-time data extract into a dynamic collection that is periodically updated and optimized for use by multiple teams and projects.

Data Processing

The first step in data processing may be dealing with personally identifiable information (PII). Textual data bring risks to the individual, along with the risk for identity theft and other general concerns about PII. Imprudent social media posts may lead to doxing, job termination, and other repercussions. Negative job evaluations can cause embarrassment and loss of reputation. This is a concern even when the PII is stored alongside the textual content, such as in a separate field, but sometimes the PII may be embedded within the content itself. It is not uncommon for social media posts to refer to people by name when describing something they said or did. As a result of these concerns, some of RAND's data can be stored and processed only on certain servers or using certain resources. For one project looking at military officers' job evaluations, the records were stored as images, making it impossible to separate the data from the PII at the top of the forms until after the forms were processed. We went through a lengthy process of digitizing the forms on an air-gapped computer to protect the PII. This process took many iterations to make sure that it was working well on the older, lower-quality forms.

Simple text processing, such as term frequency–inverse document frequency (TFIDF) or even just bags of words, often work quite well for text classification tasks. Text in officer performance bullets is actually nowhere close to "natural language." Each bullet must fit on one line, and abbreviations, acronyms, and word fragments are all allowed. As a result, our exploration of open-source NLP tools generally did not identify any that would help in this application. Instead, we found that a basic TFIDF model performed as well as or better than more-complex approaches, and it had the advantage of being easier to explain to the audience and more understandable. In the context of more-natural language, we have often improved models by excluding *stop words* and stemming words prior to constructing the TFIDF. In some cases in which extensive jargon is used, we have not seen improvements from doing this—or have even lost information—because, in jargon language, different grammatical usages of the same word can indicate radically different meanings. These simpler techniques are less processing intensive and can provide more-interpretable results (which also depend on what type of model is used).

Word embeddings may disappoint. Pretrained word embeddings using algorithms—such as GloVe, Word2Vec, or FastText—are impressive in their design and ability to solve analogies or find similar terms. In practice, they may

Because word embeddings perform well for a simple task, such as text retrieval, it may still make good sense to build customized ones, and there is considerable flexibility in doing so.

work quite well for a simple NLP task, such as text retrieval on the basis of search terms. But even for text classification and similar tasks, their usefulness has proven to be limited. Pretrained embeddings on large corpora, such as Wikipedia or the Brown University Standard Corpus of Present-Day American English, have been too generic, even in a study of tweets about health information. Another option is to train custom embeddings using a topic-specific corpus. For some projects, we have found that this improves model performance; for others, we have not. A study of military readiness built custom word embeddings using 20 years of publications from RAND's research divisions supporting DoD (including the U.S. Army, Air Force, Office of the Secretary of Defense, the Joint Staff, and combatant commands). These embeddings vastly outperformed pretrained general-corpus embeddings on analogy and word-similarity tasks but did not improve performance on a text classification task. Nonetheless, because word embeddings perform well for a simple task, such as text retrieval, it may still make good sense to build customized ones, and there is considerable flexibility in doing so. It may be particularly useful to build bigrams and other n-grams, such as *Air_Force*, *Department_of_Defense*, and *collective_training*. This definitely improved the performance of our custom embeddings on analogy and word-similarity tasks. Another option available for custom embeddings is the size of the dictionary and the minimum number of occurrences of a single word to include it. With a small corpus such as ours, we found that setting the occurrence threshold fairly low—three to five—struck the right balance.

Besides pretrained word embeddings, other tools for text processing are also available, and many can be customized. These include tokenizers, stemmers, lemmatizers, part-of-speech taggers, HTML parsers, stop-word lists, and entity extractors. But as with word embeddings, these other tools may need to be customized to the particular task. For example, many tokenizers ignore or separate nonalphanumeric symbols, so hashtags and mentions can be lost from social media data. Natural Language Toolkit (NLTK) has something called a tweet tokenizer, and spaCy tokenizers can be extended to find hashtags. It is important to remember that lemmas, like word embeddings, are based on general language usage and may have limited utility, or else will need to be customized, for topics with specialized language. Stop words are another tool to be wielded with caution. There is no canonical list of stop words for the English language, and they may need to be customized for niche applications. Also, stop-word extraction may improve models using a bag of words or some other method in which word order is lost, but with models that preserve word order, negation (*does* versus *does not*) is critical. Entity extractors can also be customized, provided that you have or can create training data. In examining news stories about mass shootings, we realized that it would be helpful to train an entity recognizer (we are using spaCy's built-in tags) on specific facts from cases, such as *weapon*, *witness*, and *victim*. A caveat is that extensive tools are available for English and Chinese, but fewer are available for other languages.

Minimizing the custom modification of data to put them into a common data processing pipeline supports repeatability across data sources. When working on a project analyzing social media content, we used data from multiple platforms. Regardless of the platform, APIs typically provide data in schema-less JSON files. With a schema-less database, even the observations from a single platform could have different fields, and there was great variation across platforms, but we needed to perform identical analysis of all the data. One option would have been to build a custom pipeline for each platform from start to finish. This would have enabled us to put multiple text processing steps into single user-defined functions, or even single lines of code, but this approach also would have made it more difficult to ensure repeatability across different data sources. Instead, we minimized the amount of custom code we had to write for each platform to get it all into a common, semiraw format, then fed the various files through a single processing pipeline. As new data sources are added, we write the minimum amount of custom code for each particular source to prepare it for the format that the pipeline is expecting. This principle applies to the output of the pipeline as well: Data should be indexed and stored such that they can be merged or used with other data sources.

Text processing can have a significant effect on model quality, especially when dealing with imbalanced data sets. Data modelers, who may be different from the data engineers, may want to experiment with multiple transformations of the data. Even data transformation may have hyperparameters that need to be optimized, such as sensitivity to false positives versus false negatives in determining which data to use for a model. For example, a text classification task looking for research on certain types of health services found that model accuracy increased significantly by setting thresholds for word-count inclusion based not only on the entire corpus but also on words' frequency in only positive and only negative observations. This step did not add many words to the entire corpus dictionary (just a few hundred) and ensured that the model observed words that were highly discriminating but occurred frequently in only one category or the other. This seems most important when dealing with imbalanced data sets, in which relatively few texts are of a particular class.

Ground truth is not necessarily true. There is a presumption in artificial intelligence (AI) that human-labeled data are the ground truth and are therefore useful for training machine learning algorithms. Different model algorithms, architectures, and hyperparameters are tested to score as closely as possible to how humans label data. But examining the data—either in advance or in specific cases that the model got wrong—may reveal that some data are mislabeled. Whether it is worthwhile or appropriate to retrain models on relabeled data is debatable and very case specific. For one text classification task, we had a U.S. Navy surface warfare subject-matter expert manually label 100 observations, and we compared his labels with those of the model and with the original source. Because the model performed reasonably well, with an F1 score (for precision and recall) of nearly 80 percent, we heavily oversampled observations that the model got wrong. In roughly half of the "erroneous" model results, our subject-matter expert agreed with the model and not with the original source. This was an exploratory project, so we went no further with the modeling, but if we were building a production system, we might have retrained the model using relabeled data or else excluded the observations we deemed to be mislabeled.

Text processing should be specific to the research question and the model. In building a model to identify Russian trolls manipulating public opinion via social media, we were confronted with determining the appropriate unit of analysis. Is it an individual tweet, is it the entire history of an individual's tweets, or is it something in between? We ended up aggregating individuals' tweets into roughly 1,000-token chunks, using each chunk as a separate observation, drawing from prior linguistic research that found that differences in people's styles of discourse begin to reveal themselves at around 1,000 words. On another social media project, the goal was to find emergent words and phrases that appear with known words and phrases of white supremacists on message boards. Again, we were faced with the question of whether to aggregate the text and, if so, how. We could either treat each post as a separate observation or aggregate on the basis of the person or on the basis of the discussion thread. We felt that it made more sense to aggregate on the basis of the thread, but not all social media are organized around threads. We ended up aggregating on the basis of the thread where we could but on the basis of the user where we could not. Because this was an unsupervised learning model that simply looked for collocated terms, that approach seemed acceptable, but it likely would not have worked if we had a supervised learning model that used training data that had been aggregated either one way or the other.

Model Development

Computing infrastructure may constrain the types of models that can be used. Modeling libraries are more difficult to install, update, and use in restricted computing environments, such as classified computing systems or on air-gapped machines for handling other sensitive data. One solution is to create a virtual environment with all necessary libraries on a networked machine and then transfer the virtual environment and conduct the modeling on the restricted machine. For this approach to work, the virtual environment must fit on a portable storage device. We were able to use this process to employ TensorFlow on a classified computer but could not use a state-of-the-art attention-based language model, such as XLNet or BERT (Bidirectional Encoder Representations from Transformers), because the parameters would not fit on our available portable storage, so we used simpler word embeddings for our model. On a different project, we were working with a DoD sponsor to build a data dashboard and received mixed messages about which programming languages and libraries are available for DoD computer systems. Besides creating virtual environments, another option would be complete containerization of the modeling tools. We did not attempt this with the DoD sponsor but have the capability ourselves to conduct data processing and modeling in containers running on AWS instances.

Graphics processing units (GPUs) are increasingly necessary for large modeling efforts. For the project on the air-gapped computer, it might have been infeasible to use the attention-based language models as the embedding layer of the text even if they had fit on portable storage. This is because, even with simpler word embeddings, some model architectures took hours to run on CPUs, and we did not have access to GPUs in a classified environment. One of the challenges of creating a bespoke NLP model is the time it takes to train. Using a GPU greatly speeds up this process, allowing for more iterations to find and fix problems. Unfortunately, GPU-equipped machines are much more expensive than CPU-only machines. With data and models that can be run in more-permissive environments, cloud computing is an excellent option. Current cloud vendors offer these machines for between \$650 and \$9,000 per month, depending on the type of GPU and whether multiple GPUs are required to fit the training data in memory. Deploying a completed model for inference also benefits from being run on a GPU. However, cloud providers have few autoscaling features that take GPU use into account, which results in either overprovisioning cloud resources to meet peak demand, or overutilizing resources, resulting in a slower end-user experience. Fine-tuning models requires more-powerful hardware (multiple GPUs in parallel) than does serving a trained model.

For many text classification tasks, simpler models often perform quite well and are easier to interpret than are more-complex models. Support vector machines (SVMs) can be very useful, although a properly fitted neural network may perform somewhat better on such metrics as accuracy or ROC-AUC (receiver operating characteristic–area under the curve; maybe three to nine points). The difference is that once all the data collection and processing are complete, it takes a suitably skilled analyst a few days to run an SVM, whereas a neural network often takes a few weeks to get right and to get through several rounds of computation time. Other machine learning algorithms—such as random forests, logistic regression (LR) models, and Bayesian networks—can also perform satisfactorily, depending on the goals of the project and the complexity of the task classification. An advantage of a model that applies a single parameter to each input, such as an LR model, is that it is readily transparent how each term contributes to the probabilities calculated for each observation.

More-challenging text classification may require a neural network to generate satisfactory results. This was apparent in a project predicting military readiness according to commanders' comments. When the outcome measure was the three-tier scale of yes, qualified yes, or no, an LR model worked well. When we tried to apply the same model (and the same text processing steps) to a different data set with a four-tier scale, it performed poorly. Admittedly, the data set and military service were different, but even with the scale used for the first project, the model was least accurate in predicting the middle tier, the qualified yeses. The second, four-tier, project essentially split the middle tier into two levels, which complicated the prediction even more. A neural network with recurrent memory units greatly outperformed the LR model. When

NLP models, probably more than any other modern branch of AI, are intended to replicate human activity. NLP models may benefit significantly from human review of the data and model predictions, particularly false positives and false negatives.

a neural network is called for, we have found that stacking multiple layers within a neural network may be very useful for image data but does not produce much of a gain for NLP models. Instead, the superior architecture with recurrent neural networks for NLP has been fewer layers with many nodes per layer.

Classification task thresholds should be adjusted to attenuate the rate of false positives and false negatives as necessary for the particular application. For example, when using NLP and supervised machine learning classification to produce population-level estimates, such as the proportion of Twitter users in country *X* with positive versus negative sentiments toward the U.S. military, classification thresholds should be set to balance the number of false-positive identifications against an equal number of false-negative identifications. In other cases, thresholds should be set to minimize false positives even at the expense of increasing false negatives, or vice versa, depending on the relative risk associated with each error type.

NLP models are especially amenable to human review. Natural language is inherently intended for human reception, interpretation, and understanding. Therefore, NLP models, probably more than any other modern branch of AI, are intended to replicate human activity. Both the inputs and the outputs of an NLP model should be understandable to a real person. When a subject-matter expert reviewed a sample of false positives and false negatives on a project predicting military unit readiness, he found cases in which he thought that the original label, not the model result, was erroneous ("ground truth is not always true"). When a modeler reviewed a sample of false positives and false negatives on a project predicting officers selected for command, he realized that certain words were highly discriminating for true positives but did not occur frequently enough in the overall corpus to be included in the original models ("ELT may improve model performance"). Besides fine-tuning hyperparameters, increasing model complexity, training for more epochs, and taking the other standard approaches for improving model performance typical of many types of machine learning tasks, NLP models may benefit significantly from human review of the data and model predictions, particularly false positives and false negatives.

Application Development and Deployment

Not all NLP projects have a true modeling stage; some basically go from data processing to application development. Massive state-of-the-art language models can often perform remarkably well on NLU tasks in their existing state without any task-specific fine-tuning. Fine-tuning may improve model performance but may also come at a high cost in terms of human time and computational resources. Thus, depending on the nature of the task, once data have been collected and processed, the next step may very well be application development. For example, on the project extracting information from news stories about mass shootings, we already had available to us not just the pretrained language models but even open-source APIs (Farm and Haystack) to ask questions of and get answers from the corpus. Given constraints on time and resources, we did not fine-tune the models and went straight to building a simple web interface (Streamlit and Flask are two good development options) and deploying it to a Docker container on AWS that multiple teams could use. When summarizing cloud vendor responses to a request for information from a federal agency, we used open-source summarization tools and focused on building a user-friendly interface on top of them.

In the application stage of a text classification task, it can be more useful to report category probabilities than to report the predicted category. Not only are models fallible, but people are right to be wary of trusting them too much—even high-performing models. Reporting to an end user (who in many cases was not the developer) that a certain observation is true, false, positive, negative, a person, a troll, and so on may elicit mistrust that discourages interaction with the tool. This seemed a likely risk as we thought about an application that would incorporate our model of military readiness based on commander comments. A chatbot reporting the model's results would likely have a much more favorable interaction with a unit commander by saying, "according to your comments, there is a 60 percent chance your unit is at readiness level three" than by saying, "according to your comments, your unit is at readiness level three." In cases of a multicategory text classification task, reporting probabilities is even more important because for any given observation, no probability might exceed 50 percent.

Meta Observations

The following observations do not fall into any one category of the data science steps outlined above and are offered as broad observations derived from multiple efforts.

Privacy, security, and licensing considerations need to be addressed at all stages. Even after data have been collected and appropriately safeguarded, the results of the data processing (such as word embeddings) or modeling (such as language models) steps may still contain people's names, other PII, or terminology that must be protected. We were somewhat surprised to find that even unit readiness reports sometimes contain the names of personnel. As for licenses, although many of the tools are open source, the data often are not, and downloading massive amounts of data may violate terms of a subscription or site license.

Even open-source tools vary in the terms users must adhere to, particularly if something is being distributed to end users.

There may be high latent demand for NLP tools. NLP tasks tend to be "upstream" activities that produce outputs for additional analysis. For example, the question-answering tool was designed to help our own analysts build profiles of events and suspects, and the analysts need to know specific facts from a case, such as whether the suspect knew his victims or had a prior arrest record. A text classification tool that categorized job openings was used to evaluate relative demand for various skills, which in turn was an input to yet another analytical task. NLP tools may have a high latent demand, in the sense that activities that are already being performed as part of a larger task or project could be done more quickly, more accurately, or at a greater scale through the use of NLP tools, but the people performing those activities might not be aware of available NLP tools or are incapable of wielding them.

Algorithms might "learn" unfairness from the training data but could also improve fairness of an underlying process. We have all been taught to be wary of machine learning algorithms doing things we do not want them to do, but in the case of our analysis of language used in evaluations of military officers, (1) research has shown that the promotion system is basically fair with respect to race/ethnicity and gender, (2) the information in an officer's records is tightly regulated and is included explicitly for the purpose of evaluating performance, and (3) human judges have idiosyncrasies and can make errors for various reasons. Thus, using a well-calibrated machine learning tool in the process of understanding officer performance will not necessarily treat people unfairly and can even enhance the fairness of the system. During this project, we recommended some additional implementation strategies to further mitigate concerns, such as incorporating randomness into the process of flagging records for manual review.

Hyperfocus on the tactical level ignores broader perspective. Across DoD, text analysis efforts (usually within larger operations in the information environment using social media) are generally limited to the individual level: a specific message or actor. This is certainly important, particularly from a force protection perspective, but it also restricts NLP efforts to the purely tactical level. It ignores an entire operational and strategic layer: whole discourses, speech communities with specific language

Activities that are already being performed as part of a larger task or project could be done more quickly, more accurately, or at a greater scale through the use of NLP tools.

Enhancing pretrained language models with tags indicating how the language was being used improved model performance and interpretability.

norms, and public or advocacy groups with shared ends or ways of speaking. In addition to considering actors within interpersonal networks, it is important to look at networks of communities and metacommunities.

NLP tools are better at determining what language is about than at determining how it is used. Existing tools are quite good at entity extraction, content analysis, and text classification. They are generally not as good at detecting sarcasm; identifying complex affect, such as regret; or analyzing rhetorical patterns or tactics. In our analysis of social media activity by possible foreign-sponsored trolls, enhancing pretrained language models with tags indicating how the language was being used improved model performance and interpretability.

NLP questions are largely treated as engineering challenges with little room for theory. Computer science (CS) approaches tend toward CS-only solutions, and NLP practitioners have a certain level of "data have made theory irrelevant" buy-in. Emergent grammar theories that treat language structure as dynamic and socially negotiated emergences have very fruitfully informed NLP and text-mining work. At least for the moment, good theory still has something to contribute to NLP. NLP practitioners from CS backgrounds should collaborate in cross-disciplinary teams and be open to useful insights around method combinations, units of analysis, and paradigms for analysis and detection. At the same time, domain experts need to learn more about CS methods and be able to fruitfully collaborate with CS NLP practitioners, not guard turf, and be open to machine and human-machine approaches.

Selected Recent Published RAND Works Employing Natural Language Processing

Jaycocks, Amber, *Climate Finance and Green Bond Evolution: Informing Policy with Machine Learning Text Analytics*, dissertation, Pardee RAND Graduate School, Santa Monica, Calif.: RAND Corporation, RGSD-A678-1, 2020. As of May 20, 2021: https://www.rand.org/pubs/rgs_dissertations/RGSDA678-1.html

Marcellino, William, Kate Cox, Katerina Galai, Linda Slapakova, Amber Jaycocks, and Ruth Harris, *Human-Machine Detection of Online-Based Malign Information*, Santa Monica, Calif.: RAND Corporation, RR-A519-1, 2020. As of May 26, 2021: https://www.rand.org/pubs/research_reports/RRA519-1.html

Marcellino, William, Todd C. Helmus, Joshua Kerrigan, Hilary Reininger, Rouslan I. Karimov, and Rebecca Ann Lawrence, *Detecting Conspiracy Theories on Social Media: Improving Machine Learning to Detect and Understand Online Conspiracy Theories*, Santa Monica, Calif.: RAND Corporation, RR-A676-1, 2021. As of May 25, 2021: https://www.rand.org/pubs/research_reports/RRA676-1.html

Marcellino, William, Christian Johnson, Marek N. Posard, and Todd C. Helmus, *Foreign Interference in the 2020 Election: Tools for Detecting Online Election Interference*, Santa Monica, Calif.: RAND Corporation, RR-A704-2, 2020. As of May 26, 2021: https://www.rand.org/pubs/research_reports/RRA704-2.html

Schirmer, Peter, and Jasmine Léveillé, *AI Tools for Military Readiness*, Santa Monica, Calif.: RAND Corporation, RR-A449-1, forthcoming.

Schulker, David, Nelson Lim, Luke J. Matthews, Geoffrey E. Grimm, Anthony Lawrence, and Perry Shameem Firoz, *Can Artificial Intelligence Help Improve Air Force Talent Management? An Exploratory Application*, Santa Monica, Calif.: RAND Corporation, RR-A812-1, 2021. As of May 21, 2021: https://www.rand.org/pubs/research_reports/RRA812-1.html

About the Authors

Peter Schirmer is a research programmer at RAND whose NLP work includes military readiness, election interference, white nationalism, and cloud computing solutions. He has an M.B.A. and an M.P.P.

Amber Jaycocks is a machine learning research scientist and an adjunct researcher at RAND. Her NLP work focuses on unsupervised text summarization across policy domains. She has a Ph.D. in policy analysis.

Sean Mann is a policy analyst at RAND specializing in data science and mixed methods research. He has used NLP to analyze research abstracts, federal program documents, and gray literature. Mann has an M.S. in foreign service.

William Marcellino is a senior behavioral scientist at RAND. He has applied NLP approaches to topics of election security, military officer talent, countering violent extremism, and operations in the information environment. He has a Ph.D. in rhetoric.

Luke J. Matthews is a senior social scientist at RAND. He has applied NLP tools to data from Twitter, military unit readiness systems, Army intelligence reports, and Air Force officer-promotion records. He has a Ph.D. in Anthropology.

John David Parsons is an application developer at RAND. He specializes in enterprise software and leveraging NLP for business intelligence and market analysis. Parsons has a B.A. in computer science and in international relations.

David Schulker is a policy researcher at RAND. His NLP work focuses on the design and implementation of performance management systems in DoD. He has a Ph.D. in policy analysis.

About This Perspective

This reference document offers a collection of lessons learned by practitioners from RAND Corporation projects that employed natural language processing (NLP) tools and methods. *NLP* is an umbrella term for the range of tools and methods that enable computers to analyze human language. The descriptions of lessons learned are organized around four steps: data collection, data processing (i.e., NLP-specific text processing in preparation for modeling), modeling, and application development and deployment. The research reported here was completed in February 2021 and underwent security review with the sponsor and the Defense Office of Prepublication and Security Review before public release.

RAND National Security Research Division

This Perspective was prepared at the request of the U.S. Department of Defense's Joint Artificial Intelligence Center and is based on the practical experience of the authors, who have worked on a variety of projects that included natural language processing tasks. The effort was conducted within the Acquisition and Technology Policy Center of the RAND National Security Research Division (NSRD). NSRD conducts research and analysis for the office of the Secretary of Defense, the U.S. Intelligence Community, U.S. State Department, allied foreign governments, and foundations.

For more information on the Acquisition and Technology Policy Center, see www.rand.org/nsrd/atp or contact the director (contact information is provided on the webpage).

RAND CORPORATION

www.rand.org

PE-A926-1