//// How Fireside is Leveraging Machine Learning to Improve the US Congress

Joseph Nelson, Scott Sadlo, Isabella Seeman

Intro

The US Congress receives over 10 million messages a year - sometimes to the tune of 300 messages per minute. Many trends contribute to an increasingly engaged citizenry and busy legislative branch over the last decade: technologies have simplified emailing and calling Congress, the average number of people per Congressperson is ever increasing (now over 609,000 to 1), and social media provides an entirely new channel to reach elected officials.

While an increased citizenry is a decidedly positive facet of democracy, Congress's ability to handle a rapid growth in citizen messages has not kept pace. At Fireside, we're determined to improve relationships between citizens and the US Congress to empower effective governance. For the same reason technology makes it easier to be an engaged citizen, we're squarely focused on leveraging it to improve the US Congress's ability to keep to serve US citizens.

The advent of natural language processing, AI, and machine learning, in particular, have created new opportunities to empower the US Congress to scale its workload demonstrably. In the same way that companies like Zendesk assist organizations like the Salvation Army with responding to inbound requests for assistance, Fireside21 can create tools that enable hard-working Capitol Hill staff to serve more constituents, faster.

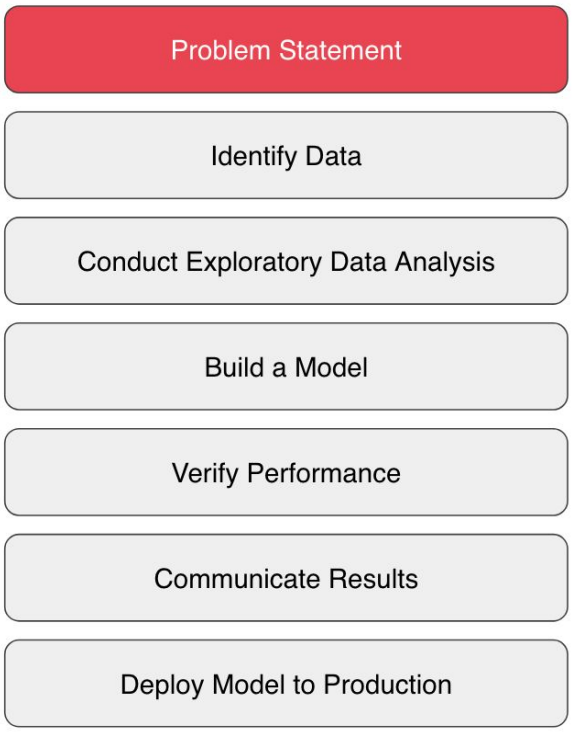
Machine learning, specifically, is a branch of artificial intelligence that engages in the practice of creating systems that allow algorithms (a set of rules) to identify what outputs to provide based on a series of inputs. Perhaps more simply, machine learning is showing computers how to identify patterns — sometimes patterns that humans may have otherwise missed. For example, Netflix's movie recommendation system learns patterns based on how its users interact with various media. If User A rated "The Office" 4 / 5 and "30 Rock" 5/5 and User B rated "30 Rock" 4 / 5, the recommendation engine may recommend User B watch "The Office."

Relatedly, natural language processing is a branch of machine learning that allows computers to understand written language more closely to how humans do. Language, as written, is "unstructured data." For example, this post would not fit neatly into an Excel spreadsheet. Instead, we would need to process, say, the count of each words to turn the text data into numeric, tabular information. Once text is structured data, we can mine it for insights, identify topics, and even predict what words someone may use next.

Taken together, machine learning and natural language processing enable impressive technological feats. Fireside21 envisions a world where technology extends the capabilities of Congress: quicker identification of emerging constituent needs, faster sorting of inbound messages, deeper and more custom letter responses, and more. The below detailed process has just scratched the surface of improving constituent and US Congress relationships with machine learning.

A Deep Dive into Fireside's Machine Learning Process

Data science and machine learning problems follow a common flow: clearly define a problem statement, collect relevant data, conduct exploratory data analysis, build a model to predict the outcomes, verify performance, communicate results, and place the model into a production environment (include the model as a feature). Often, this process is highly iterative where results from a downstream process impact prior decisioning.



Problem Statement Identification

Given the expansive impact machine learning and natural language processing can have on Fireside21's product suite, identify *a* problem to solve is not a challenge, but rather prioritizing an option that is both *high impact* and *ease of implementation* is key.

As the team considered ways to incorporate data science into the product, the following areas were discussed:

- Surfacing office productivity analytics
- Offering a District "Pulse Check" dashboard
- Auto-tagging and sorting inbound of constituent letters
- Creating a shared cross-office letter library
- Surfacing relevant articles mentioning a member of Congress
- Enhancing a constituent profile directory
- Developing social listening (social media mentions, sentiment) for offices

While not all opportunities necessarily require machine learning, all may benefit from a layer of said technology. Fireside21 considered areas that simply would not be possible without the advent of machine learning. That is, features that may previously have been

impossible or significantly hindered if machine learning were not incorporated were prioritized.

Auto-tagging and sorting of inbound constituent letters stands out as a clear high value add opportunity for machine learning. Absent ML, letters must be painstakingly manually sorted, often resulting in weekends where Capitol Hill staffers work through backlogs. Without ML, at best, exact text matching techniques to catch inbound form letters (duplicate message content from different senders). Moreover, freeing staffer time from sorting messages creates greater opportunities to craft custom, specific responses to individual correspondents. The more representative democracy feels responsive and personal, the better the relationship is among citizens and member.

With a domain identified, the Fireside21 team set a specific goal:

> *Can we create an inbound message classification system that predicts an inbound message's expected office response with at least 80 percent accuracy.*

This goal requires context and caveats. For machine learning to be effective, we need to train an algorithm on what is a correct vs inaccurate prediction. In the case of receiving an inbound message, the most objective way to determine a system has understood its contents correctly is to prove the system made the same decision a person would have. That means matching an inbound letter to a response. Critically, Fireside21 aims to encourage offices to have a high number of potential responses. If, hypothetically, an office responded to every inbound letter with the same generic response, a system that recommends replying with said letter regardless of the letter's context would be 100 percent accurate. Thus, the goal is not *just* to prove a message can be sorted correctly. A successful system, when put into production, will not cause an office to reduce its number of potential responses. Ideally, the opposite occurs with less time spent sorting and more time spent responding.

Identifying Data

Given the team aimed to create a machine learning system that correctly classified (sorted) inbound constituent letters and emails, the data sources that follow are clear: example inbound messages and offices responses. Digging deeper, however, Fireside21 needed to identify offices that may be good options for researching and developing this intelligent recommendation engine. Offices have varied amounts of letters, potential responses, geographic influences on the content of responses, and much more. (Anything, in theory, that affects the political temperature of the United States affects the datasets selected.)

Fireside21 determined using three clients of varied geography and, critically, message response habits would allow clear comparison and contrast of technological techniques on the recommendation engine's performance. Inbound message data from quarters three and four of 2017 were ultimately selected. It is worth noting this data is raw: errors (like null letters), incongruencies, duplicates may all be possible. Handling this situations is key in the exploratory data analysis phase is critical.

Exploratory Data Analysis

Exploratory data analysis (EDA) refers to the process of cleaning, wrangling, processing, visualizing, and using statistical inference to better understand our data. For example, simple counts of how many letters were sent, identifying null and duplicate letters, and identifying how long an office takes to ship a response are all forms of exploratory data analysis.

In this process, Fireside21 uncovered notable facts that aided model development.

Number of letters received, September - December 2017:

Office A: 28,720

Office B:

Office C:

Number of inbound letters whose message body was a duplicate (typically encouraged to be sent by advocacy organizations):

Office A: 11,888 (41 percent)

Office B:

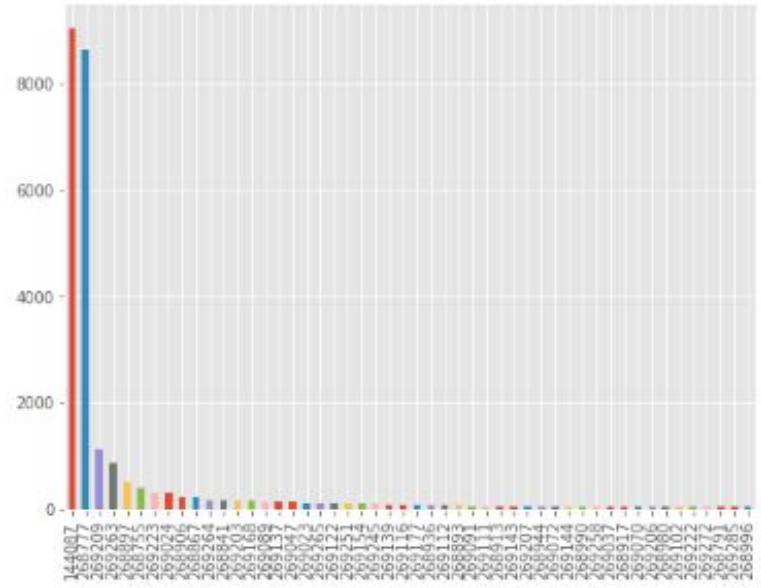
Office C:

Number of unique office letter responses utilized in the period (a proxy for customization):

Office A: 464, 176 used more than 10 times in the period

Office B:

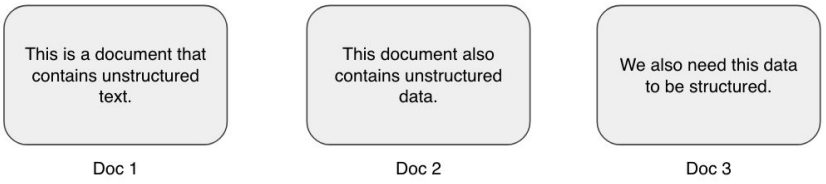
Office C:



Given office responses have such a heavy right skew (that is, letters that are used more than roughly 100 times are used a magnitude more many times), it is reasonable to apply machine learning only to inbound messages that have a sufficient number of office responses. Said another way: we want to train our machine learning model on examples. If a given response does not have enough examples that our model can learn from, we lack certainty that our model is learning at all. For this reason, our EDA informed that we should only include responses that have been used in excess of ten times for training.

The message data, which is text, also needed to be converted to structured, numeric data for algorithm to identify noise. Computers, generally, require data to be in spreadsheets for the purposes of analysis. Yet text data is not spreadsheet-friendly absent preprocessing.

A relatively naive, but effective, way to convert text data into structured data is to count words used across a document and store the results in a spreadsheet. This is known as “count vectorization,” as we count the number of occurrences of each word across documents and store the results in vectors (structured arrays of numbers).



	this	is	a	document	that	contains	unstructured	text	also	data	we	need	to	be	structured
Doc 1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Doc 2	1	0	0	1	0	1	1	0	1	1	0	0	0	0	0
Doc 3	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1

In its simplest form, count vectorization treats all words as independent data points to count. At this stage, We can make decisions that optimize the signal we extract from this text-to-numeric transition. For example, words like “a,” “the,” and “also” may add little signal to a message’s contents. We can remove them from the dataset. Moreover, counting each word in independence may cause us to miss important terms like “White House,” so we can tell our vectorizer to count single and double word occurrences each. Lastly, we only want to include terms that occur within at least two documents and cap the total number of unique words we count across all documents to the 200,000 most frequently occurring terms. Note: many of these restraining factors were not arbitrarily

chosen. The Fireside21 iteratively attempted different combinations and permutations to identify which performed best for the models.

Building a Model

Machine learning relies on mapping inputs to outputs through pattern recognition. A huge benefit of leveraging algorithmic decision-making is not just that a computer may, ultimately, sort inbound messages more effectively. It is also that machines often pick up on signals that distinguish letters that humans do not. This is what headlining stories about machine learning systems outperforming humans often cover.

Before detailing each model independently, it is important to emphasize what each model is attempting to do: identify which words in a given inbound message best allow us to predict what a given office's response will be. As per our problem statement, we hope to identify and tune a model that is able to breakthrough 85 percent accuracy in this process.

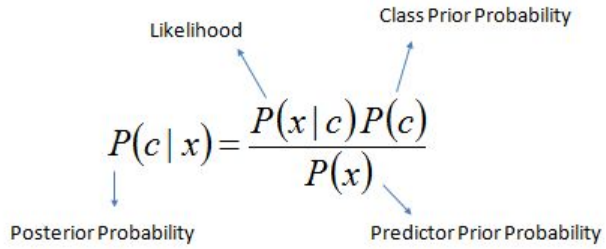
To achieve maximum performance, we attempted many different supervised machine learning techniques that mapped our inputs to our outputs. The following models were tested across the three offices:

- Naive Bayes
- Support Vector Machines
- Random Forests

Naive Bayes is a modelling technique that is based upon Bayes' Theorem, which helps us understand conditional probabilities based on evidence. For example, if someone has a lupus versus a influenza, there symptoms are quite similar for diagnosis. The broader population (the evidence, in this case) shows us that the flu is far more common (30x at least) than lupus. Given this evidence, when we see symptoms for the flu that may also be lupus, we are more likely to conclude an individual has the flu rather than lupus.

The same conditional thinking may be applied to our letter sorting task. Hypothetically, given a message has words like "border," "ports," and "people," is the letter more likely to be about immigration or climate change? Perhaps more tricky, is the letter more likely

to be about immigration or national security? In each case, we can see why disambiguation is possible.

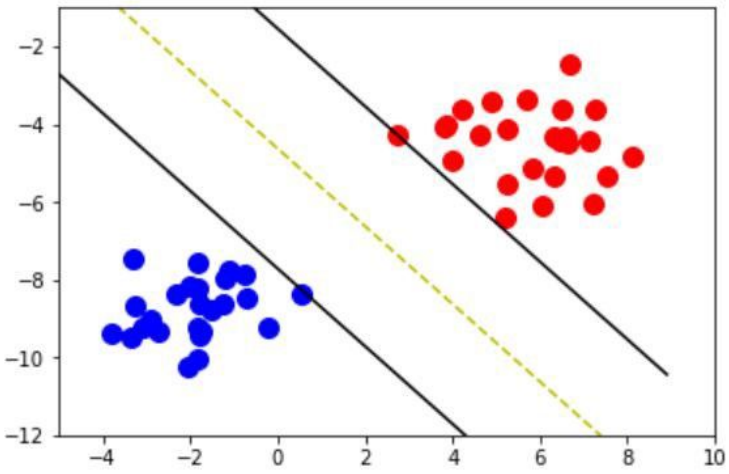


$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

credit

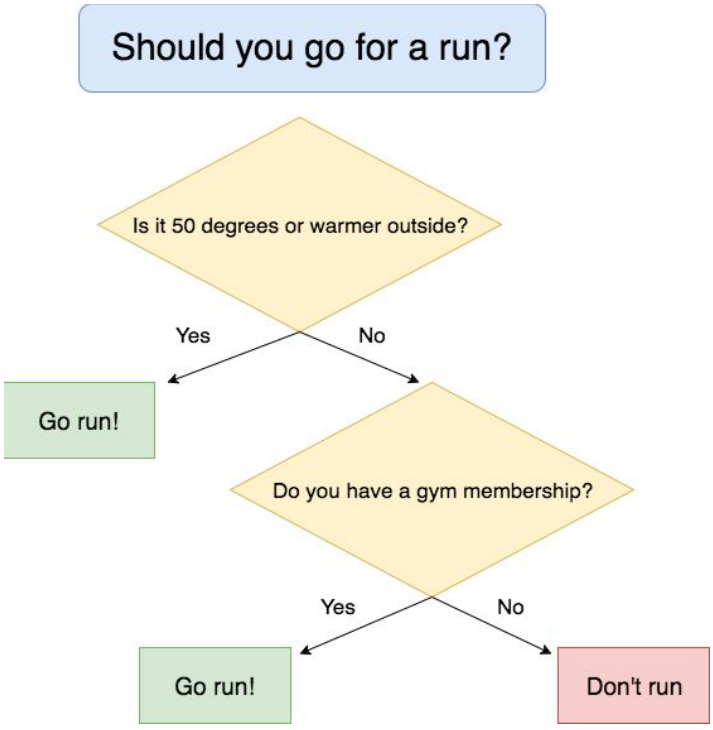
Naive Bayes models were what powered Gmail's initial spam filtering products: identifying if, based on words used, an email is spam or quality.

Support vector machines (SVMs), on the other hand, attempt to "draw lines" that clearly separate between our classes, or response letters. Imagine we were identifying if a student is likely to pass a class or not based on two attributes: class attendance and time recorded studying. If we were to plot these features, we could envision that passing students would be on the upper right portion of our graph (a high attendance and time recorded studying) while non-passing students would be in the lower left. A support vector machine attempts to draw a boundary in space between these clusters of passing and non-passing students. A boundary is well-defined if the distance between the boundary and the two clusters is high. Said another way, we want to be halfway between the two classes when separating our classes.



credit

In the case of our letters, we have many, many dimensions (200,000, in fact, as each word is a dimension) to attempt to separate. We cannot easily visualize this, but we can envision trying to separate our clusters of letters containing similar words in space.



Through research, support vector machines have proven to be strong candidates for text classification texts, such as determining if a given letter deserves a specific response.

A random forest classifier is built on foundations of decision trees. A decision tree is a classifier that acts like a game of 21 Questions where we progressively ask increasingly narrowing questions to identify an outcome. For example, if you are determining whether you should go for a run on a given day. If it is over 50

degrees, perhaps you decide you should. If it is not warm, you then ask if you have a gym membership. If yes, you run at the gym. If not, you stay home.

The way a decision tree becomes a random forest is through building many different decision trees, and averaging the best results of each. For example, perhaps the best way to determine if you should go for a run isn't by starting with temperature; it's by asking if you own running shoes. Assuming you had given the model this data (temperature, gym membership, running shoes), one decision tree could be built with the first question asking if you have running shoes. If this, on balance, produces better results than other decision trees, we will keep the first question as identifying if you own running shoes.

In our case, we are asking the decision tree if various combinations of words are present in a given letter. If they are, we follow that according branch. If they are not, we go to the other branch. Ultimately, we arrive at a terminal leaf that tells us which response was most likely used for that given letter.

Random forests are strong performers for text data given decision trees capture non-parametric features. In other words, if words across letters are occurring with substantially different frequencies (as they are), a random forest does a powerful job of picking up on these irregularities to create accurate results.

Ultimately, results showed that Naive Bayes classifiers performed best in each case regardless of office. This is not terribly surprising: Naive Bayes models are known to perform quite well on text data.

Congressperson	**Classes**	**Lift**	**Accuracy**	**Best Model**
Office A	4	0.12	0.79	Naive Bayes
Office B	21	0.39	0.70	Naive Bayes
Office C	15	0.31	0.91	Naive Bayes

Validating Model Performance

Each model must be rigorously tested to assure it is making inferences that we find explain the data well. Absent validating the model can generalize to other situations, we would not be confident the model will work well in production (as a feature in the Fireside21 product).

To validate our models, we utilized train, test split and cross validation. Both techniques are based on a simple premise: given we are attempting to build a model that can classify an inbound message that we have never seen before, why not pretend to show the model an inbound message it has never seen before in advance of putting the model into production. In other words, if we have 100 observations for our machine learning task, we could train the machine learning model on 80 of the examples. The last 20 are used to simulate out-of-sample observations—letters that we pretend a constituent has written in from which our model has yet to learn anything. Cross validation builds on this training and testing split in that we use different splits of data to iteratively train and test.

In our performance, we follow an 75/25 split: 75 percent of letters were used for training and 25 percent were used for testing. At this point, we also dug deeper into one specific office's data (Office A) to fine tune a model to see if we could achieve our goal of at least 80 percent accuracy in classification. In this given office, this resulted in training on 8,285 letters and testing on 2,762 letters.

With additional tuning of the Naive Bayes model and count vectorization techniques, we achieved a model performance of 80.1 percent. That is, if you provided our model with any inbound constituent letter, it would accurately predict which response letter an office would want to use with roughly 80 percent accuracy.

Unsatisfied with just narrowly achieving the 80 percent goal, Fireside21 hypothesized how the feature would be implemented in the product, and how that would impact our methods. To auto-sort a message, the product could hypothetically present the Capitol

Hill staffer with a drop down menu of the top five most likely letter responses. Thus, the model's accuracy may be assessed by not just how often the correct response is the top recommendation but among the top five. With this change in mind, accuracy achieves roughly 92 percent.

Communicating Results, Deploying to Production

The Fireside21 machine learning team is now beginning to externally communicate the results of its research and development efforts, starting with this post.

Given the team has produced a model that is of satisfactory performance in a testing environment, the Fireside21 team is now ready to identify a potential pilot office for a trial implementation. While R&D efforts have proven promising, deployment to a production environment will require tackling new challenges like regularity of model retraining and assuring the models have necessary compute power to perform successfully.

Summary

Fireside21 aims to improve constituent and Congressional relationships through technology products. The advent of machine learning and natural language processing presents a unique frontier to deliver products unlike any previously that not only scale a budget- and time-constrained office's abilities, but provide a stepwise improvement in their understanding of their district. While efforts, thus far, have focused on automatic message classification to reduce time spent sorting, thereby reducing time to response and increasing staffer time on quality of response, the team has only just begun to research and develop broad machine learning potential. The team looks forward to productionizing satisfactory inbound message classifications as well as exploring new verticals to which machine learning will be applied.

Research Notebook