**Introduction: What is the purpose of the project?  What problem are you solving?**

Political polarization in America has become a major problem for our democracy. The Constitutional Republic was designed to have states, parties, and citizens with different desires work together and accept compromises. In George Washington’s farewell address, he warns of dangers of political factions and advised citizens to view themselves as a cohesive unit.

The polarization in America has shown its negative effect in civil and social life. Political deadlock in Washington is a result of polarization because any hint of compromise is seen as concession to the despicable “other side” in a fight between good versus evil. Social life is rife with innocuous statements by ordinary citizens being interpreted as “dog whistles” that support the opposing side. Calls to violence have become more frequent with memes like “punch nazis” being popularized amongst a radicals. The Overton window on what is acceptable treatment of individuals that do not share identical values is shifting to become ever more nasty and violent.

With very little effort each one of us can come up with a partisan message that is perceived positively by one side and negatively perceived by the other. The epidemic of fake news is an instance of this problem, where foreign actors with very little cultural knowledge of the US can generate extremely divisive messages and spread them easily on social media.

Coming up with a message that can be well received from people on both sides of the political spectrum is a much more difficult task. The goal of my application is to allow people to discover those messages on Twitter. With the hope that those messages can be promoted and spread in a way that reminds fellow Americans that we still have things in common despite our political differences. Another use of my application is to send a tweet, measure it’s polarization, and adjust the message to better appeal to all Americans.

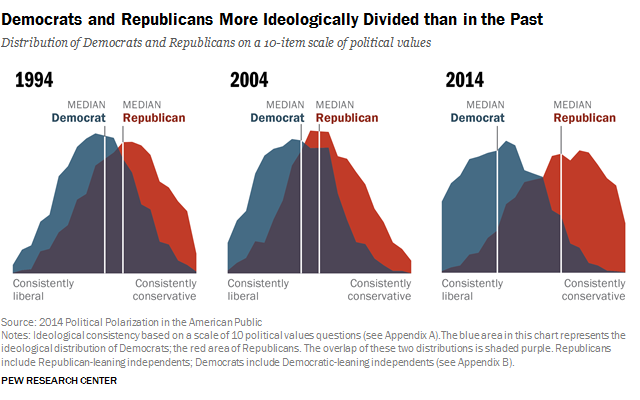
**Background:  What's wrong with existing solutions?  What's unique/interesting about this problem?  What design choices did you make while planning your solution?**

There are a number of research papers that use a variety of methods to try to measure polarization of a party, general electorate, or even of users on Twitter. But there doesn’t seem to be any technical solution with a user interface that can give people up to date information on polarization of specific messages.

What I found interesting with this problem is how best to measure this polarization. My first thought was something like a sentiment analysis on how polarized the wording or phrasing of a particular text. The problem that I discovered is it is very difficult to get labeled data for how polarized a text is. Even if done manually, humans (myself in this case) are not great at predicting what is liked/disliked by others due to our inherent biases. That’s why I decided to use a more distributed approach to my measurement. If I can figure out who agrees/disagrees with a text and which way these people lean politically, I can determine the polarization of a text.

The other major decision I had to choose was what was type of text am I going to label here? The contenders were Facebook, Twitter, and Reddit. Facebook data is more private than the other two. Facebook users can have long political rants, which is exactly what I’m looking for, but this is only accessible to people who are friends with them. Reddit, like Facebook, has a lot of text that I can use. Their data is publicly available as well. However, the developer API to access this data isn’t very open. Reddit is also not often used by public figures. I knew that I needed to at least initially get some good labeled data. Twitter is great for this, every political representative or political pundit has a Twitter account. Twitter’s developer API is also very feature rich (but rate limited) which would make the collection of data much easier.

Finally, I needed to be able to determine the political leanings of a Twitter user. I attempted to look at the links between my labeled users and the user in question. However, this is where the limitations of the Twitter API blocks me. Getting complete follower/following information for even a small group of users would be virtually impossible. I had to somehow determine that someone following Ben Shapiro and Ted Cruz was different from another person who follows AOC and Ezra Klein. There is a number of studies that conclude Liberals and Conservatives use different words, often to describe the same thing. This was going to be the key to how I differentiate if a user was right-leaning or left-leaning. I would get a corpus of words used by both right-leaning and left-leaning Twitter users. If I have a user that I don’t know the political leanings of, I would compare their tweets and see if they were more similar to the right or left leaning Twitter users in my training set.

* 
* Suggestions for reducing political polarization are usually things like “intergroup contact” and “reforming the electoral process” which are fine solutions.

**Solution Description: What are the components of your software?  What tools did you use to build your solution?  How is your software organized/designed?  Diagrams would be very helpful for this section.  An appropriate level of detail would describe some of the design patterns you used and high level software organization or pseudocode for important algorithms, but including source code is probably a bit too low level.  You may want to split this into several sections/chapters/whatever.**



*High level overview of TwitterPoliticalUnifier*

Front-end:

* React + redux
* AWS Cognito

Back-end:

* Flask
* Celery for asynchronous task management
* Twitter API (tweepy library)
* Web scraping (beautiful soup)
* AWS RDS for MySQL

**Flask Web Framework**

I decided to implement a REST API with the Flask Web Framework. Flask was chosen for a number of reasons:

* Lightweight and modular
* Lots of documentation
* Fast to get up and running
* Python based

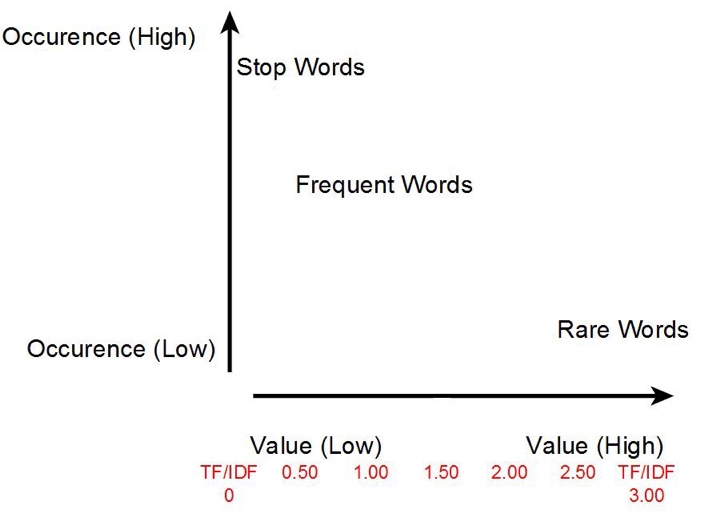
**Natural Language Processing**

To get my labeled data I used two approaches:

1. Manually gathered a list of known public conservatives (ex. Ben Shapiro, Ann Coulter ) and liberals (ex. Ezra Klein, Cenk Uygur) and democratic senators and congressmen.
2. Get the list users who follow the official @gop account and the official @TheDemocrats accounts. Remove anyone who follows both accounts. Then take a sample from each list of followers.

Once I have my labeled data I collect 200 tweets from each user. I split the sample into testing set and a training set. For each collection of tweets I do the following.

1. Noise removal
   * remove stop words, punctuation, links, hashtags, mentions etc.
2. Lexicon Normalization (Stemming)
   * a rule-based process of stripping suffixes (“ing”, “ly”, “es”) from a word
3. Create feature vectors
   * Term Frequency – Inverse Document Frequency (TF-IDF)
   * Train the feature vectors
     + Each word is given a statistical weight that measures how important a word is to a document in a corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.
   * Apply TF-IDF vectorization model on test data

* 

1. Perform classification with SVM (linear)

**User Authentication**

The scraping and analyzing of hundreds of tweets of multiple users takes a long time. There would be a delay between the user requesting a tweet or Twitter user to be labeled and the results being presented. The original idea was to send each user an email once the task was complete. After some consideration this did not feel like a great user experience. Therefore, I decided to create a user dashboard centric design to display the results of my NLP prediction. This required me to create a user profile and store information related to that user. Amazon Cognito is a tool that handles the low level details of users sign-up and sign-in. This tool handles authentication, password storage and encryption and other things like email confirmations. When a user is signs in on my app a request with their username and password is sent to Cognito. If the username and password matches, a JWT Token that holds user information is signed and returned to the client. This JWT token is stored in what react calls the state. Thereafter, every REST API request sent to the backend must be accompanied by this JWT token. The flask server receives this token from the client, verifies it against the Cognito public key. If the token is verified, the request is processed, otherwise an unauthorized response is sent back.

**Asynchronous Execution of Long running tasks**

To ensure that my API server can handle multiple requests from a multiple clients, I implemented a task queue that could handle asynchronous execution of tasks outside the HTTP request-response cycle. Celery was chosen to handle this. The normal flow of my application for the request is received, a task is started to run in the background using Celery and a response to indicate that the task has been started is sent back to the server.

**Collecting Twitter Data**

Twitter makes available an API for developers to access some twitter data. I used a popular wrapper around the Twitter Developer API called Tweepy. Tweepy wraps the low level detail of a request to the Twitter API in easy to understand and configure functions. The Tweepy is great for collecting some data, for example getting Twitter User IDs or getting the screen names of users who liked a tweet. However, the very important tasks of collecting tweets texts from each user to analyze was severely limited by Twitter. For that reason I needed to use some web scraping. The limitations of the Twitter Developer API has prompted many developers to create their own work arounds. The Iibrary that I felt best suited my needs was the Twitter-Search-API-Python. This library uses the popular web scraping framework BeautifulSoup4 to collect thousands of tweets for a specified user.

**React + Redux**

My react app stores authentication information in the top-level state of the app. Instead of passing authentication state up and down the application tree, I decided to implement Redux for state handling. Redux provides a store that can be accessed at any level of the application.

*Insert diagram of redux vs. without redux*

**Results:  What does your software do? What are it's successes and limitations?  Screenshots/a short accompanying video would be good for this section**

**Conclusions: Did you make the right design choices?  What future work could/should be done for this project.**