Sentiment Analysis on United States Presidential Speeches

Author: Dasha Asienga

Description

The project is an application project that uses machine learning to attempt to discover something new, interesting, and significant from real-world text data. Specifically, the focus is on natural language processing, with the goal of training a model to classify political speech as positive or negative based on sentiment as well as identifying patterns within certain parties and contexts. Machine learning is useful in achieving this because of the powerful classification algorithms and computing power to draw key insights on patterns and sentiments from large political speech data sets, which were previously difficult or impossible to detect. This can lead to more informed voting and policy-making.

The first step was to find data. Since the data was not labeled, the next step was to label the data by sentiment. After obtaining data, pre-processing and feature extraction were crucial steps that involved a lot of work before training a machine learning model and drawing conclusions. I assessed the performance of the models using metrics such as convergence time and accuracy rate.

In addition to running a model on the entire data set, as an alternative, I also ran other models on the data set split by era to assess whether there is some polarized sentiment or more homogenous sentiment within certain parties and/or eras. Finally, I visualized the results using word clouds to aid in the reporting of the conclusions, followed by a discussion of the implications.

My success is measured by achieving the ultimate goal of the project, which is to identify key patterns in political speech and draw key insights, potentially providing valuable implications for understanding the state of political speech in the United States on a national scale. This can provide insights into various aspects of politics and have direct impact by improving political communication and allowing politicians to tailor their messaging to be more effective and engaging with their target audience, identifying bias, propaganda, or misinformation that may be present, and improving political campaign strategies and policy-making.

Status

What I've done

I have obtained a data set of presidential speeches by all presidents in the US since 1792 until 2019. Since I could not find any meaningful data sets in this area that were labeled, I took on a lot of the work of cleaning the data and labelling it by sentiment before I could run a classification model. This involved tokenizing the 44 documents into 152,857 sentences, standardizing and pre-processing it by removing punctuation, numbers, leading and trailing spaces, and converting all text into lowercase, removing stop words such as "is", "and", "or", etc, and lemmatizing the text, that is, converting every word into its meaningful root form such as "learning" to "learn". After preparing the text, I used the VADER lexicon to classify each sentence by sentiment and chose sentiment value cutoffs to partition my text into 3 classes: positive, neutral, and negative. After this, I then used Bag of Words and TF-IDF to extract features from the text that were used as inputs into machine learning algorithms of choice. I then split the data into train and test sets, fit a few models, optimized the hyperparameters, and assessed performance. I further delved into an analysis of first party system v sixth party system to compare and contrast model performance and investigate if there are any changes over time. Finally, I visualized and reported my results.

The Data

The main dataset, `presidential_speeches.csv`, contains all official presidential speeches recorded in history from 1792 until September 9th, 2019.

There are 7 variables in the data set:

- Date - Date the speech was given
- President - President giving the speech
- Party - Political affiliation of the President
- Speech Title - Title of the speech
- Summary - An official summary of the speech
- Transcript - An official transcript of the speech
- URL - The source URL for the speech

`corpus.csv` is a condensed version of the full data set that only contains the president, political affiliation, and the respective speech.

Finally, we have separate data sets for each political era -- a political era is defined as "a model of American politics used in history and political science to periodize the political party system existing in the United States." The six subsets of the data are as follows:

- `first_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1792 to 1824.
- `second_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1828 to 1854.
- `third_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1854 to 1895.

- `fourth_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1896 to 1932.
- `fifth_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1932 to 1864.
- `sixth_party_corpus.csv` is a subset of `corpus.csv` containing all respective speeches from 1864 to present.

The data set was originally scraped from a publicly available source: https://millercenter.org/the-presidency/presidential-speeches.

More information can be found at https://www.kaggle.com/datasets/littleotter/united-states-presidential-speeches.

The original purpose of the project was to perfom sentiment analysis on political data by party, but this rich data set offers another dimension through which sentiment analysis can be performed.

Read in Data

In [31]:
```
import pandas as pd

data = pd.read_csv('COSC 247 Project/data/presidential_speeches.csv')
corpus = pd.read_csv('COSC 247 Project/data/corpus.csv')
first_party_corpus = pd.read_csv('COSC 247 Project/data/first_party_corpus.csv'
second_party_corpus = pd.read_csv('COSC 247 Project/data/second_party_corpus.cs
third_party_corpus = pd.read_csv('COSC 247 Project/data/third_party_corpus.csv'
fourth_party_corpus = pd.read_csv('COSC 247 Project/data/fourth_party_corpus.cs
fifth_party_corpus = pd.read_csv('COSC 247 Project/data/fifth_party_corpus.csv'
sixth_party_corpus = pd.read_csv('COSC 247 Project/data/sixth_party_corpus.csv'
```

In [3]:
```
corpus.head()
```

Out[3]:

	Unnamed: 0	Party	transcripts
0	George Washington	Unaffiliated	Fellow Citizens of the Senate and the House of...
1	John Adams	Federalist	When it was first perceived, in early times, t...
2	Thomas Jefferson	Democratic-Republican	FRIENDS AND FELLOW-CITIZENS, Called upon to un...
3	James Madison	Democratic-Republican	Unwilling to depart from examples of the most ...
4	James Monroe	Democratic-Republican	I should be destitute of feeling if I was not ...

Data Pre-Processing

We need to begin by cleaning and standardizing our text to prepare it for analysis.

In [4]:
```
# Select only the text column
df = corpus[['transcripts']]
```

Notice that each row is a whole body of text by the respective president. Let's split those paragraphs of texts into sentences and stack them into one column of multiple sentences.

In [5]:
```
# Tokenize text -- split each row by sentence to create multiple rows
df = (
    pd.DataFrame(corpus["transcripts"].str.split(".").tolist(), index=corpus.ir
    .stack()
    .reset_index()
    .rename(columns={0: "transcripts"})
    .drop("level_0", axis=1)
    .drop("level_1", axis=1)
)
```

We now have 152,857 rows in our data set and this is a rich data set for our intended machine learning algorithms.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152857 entries, 0 to 152856
Data columns (total 1 columns):
 #   Column       Non-Null Count   Dtype 
---  ------       --------------   ----- 
 0   transcripts  152857 non-null  object
dtypes: object(1)
memory usage: 1.2+ MB
```

Next, we need to remove punctuation and standardize our text because we are only interested in the actual words contained in the speech.

In [7]:
```
# transform each sentence into lower case
df["transcripts"] = df["transcripts"].apply(lambda x: x.lower())

# replace any non-word character with an empty string
df["transcripts"] = df['transcripts'].str.replace(r"[^\w\s]",'', regex = True)

# replace any number with an empty string
df["transcripts"] = df["transcripts"].str.replace('\d+', '', regex = True)

# remove spaces at the beginning or end of the string
df["transcripts"] = df["transcripts"].apply(lambda x: x.strip())
```

The text is now clean and in a standardized format, as seen below.

In [8]:
```
df
```

Out[8]:

	transcripts
0	fellow citizens of the senate and the house of...
1	on the one hand i was summoned by my country w...
2	on the other hand the magnitude and difficulty...
3	in this conflict of emotions all i dare aver i...
4	all i dare hope is that if in executing this t...
...	...
152852	were very much involved
152853	we very much know whats going on and were very...
152854	okay thank you all very much
152855	thank you
152856	thank you very much

152857 rows × 1 columns

In [9]:
```
#pip install nltk
```

Before performing the final sentiment classification, let's remove stop words, which can cause noise. Stop words are commonly used words like "the", "and", "of", etc. that do not carry much meaning on their own. This will allow us to retain only words that are intended to convey meaning.

In [10]:
```
# Import stopwords package from nltk
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')

# Remove the stop words
df['transcripts'] = df['transcripts'].apply(lambda x: ' '.join([word for word i
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/dasienga24/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Finally, let's lemmatize the text -- that is, let's transform all the words in the text into their proper root form, such as 'learning' to 'learn' and 'citizens' to 'citizen'.

In [11]:
```
# Import lemmatizer package from nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Create a function that splits each sentence into words, lemmatizes the words,
# original sentences
def lemmatize_words(text):
```

```
    words = text.split()
    words = [lemmatizer.lemmatize(word, pos='v') for word in words]
    words = [lemmatizer.lemmatize(word, pos='n') for word in words]
    return ' '.join(words)

# Lemmatize the sentences
df['transcripts'] = df['transcripts'].apply(lemmatize_words)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /home/dasienga24/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

The snippet below shows us that our text does not contain any more stop words and all the words are now in their meaningful root form. We are now ready to proceed with sentiment classification.

In [12]:
```
df.head()
```

Out[12]:

	transcripts
0	fellow citizen senate house representative amo...
1	one hand summon country whose voice never hear...
2	hand magnitude difficulty trust voice country ...
3	conflict emotion dare aver faithful study coll...
4	dare hope execute task much sway grateful reme...

Sentiment Pre-Classification

Because our data is not pre-labeled, before we can run any machine learning algorithm, we need to classify each text according to its sentiment: positive, neutral, or negative.

We will use a pre-trained sentiment analysis model in the Python NLTK library to label each of the sentences in the dataframe by sentiment. Note that the NLTK model may not be 100% accurate, but the data preprocessing we've done above should hopefully make the model as robust and as accurate as possible.

We are using the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon. VADER sentiment analysis relies on a dictionary which maps lexical features in words to emotion intensities called sentiment scores. The lexicon contains over 7,000 words and their associated sentiment scores, ranging from -4 (extremely negative) to +4 (extremely positive). VADER also takes into account the intensity of sentiment and the polarity shift (i.e., a change in sentiment from positive to negative or vice versa) in a given text. The sentiment score of a text can be obtained by summing up the intensity of each word in the text.

Note that there are many other lexicons in Python that we could use, such as TextBlob, but our choice of VADER is based on the fact that it is highly effective in multiple domains and one of the most popular lexicons used for sentiment analysis in Python.

In [13]:
```
from nltk.sentiment import SentimentIntensityAnalyzer

# Download the lexicon for the SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

# Initialize the Sentiment Analyzer
sia = SentimentIntensityAnalyzer()

# Define a function to get the sentiment score for each sentence
def get_sentiment_score(sentence):
    return sia.polarity_scores(sentence)['compound']

# Apply the function to the 'transcripts' column and create a new 'sentiment' c
df['score'] = df['transcripts'].apply(get_sentiment_score)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/dasienga24/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [14]:
```
df_test = df.sort_values(by=['score'])
df_test.head()
```

Out[14]:

	transcripts	score
71205	moreover judgment crime rape always punish dea...	-0.9901
32909	desire attention congress shall strictly confi...	-0.9878
320	legislative proceed unitedstates never trust r...	-0.9872
70272	abuse perhaps chief although mean one overcapi...	-0.9869
38577	attention call serious civil disturbance accom...	-0.9868

In [15]:
```
df_test.tail()
```

Out[15]:

	transcripts	score
24	thus impart sentiment awaken occasion bring u ...	0.9963
2800	may allow add gratify spectacle shall read cha...	0.9971
17439	maintain inviolate great doctrine inherent rig...	0.9980
2515	jam madison whereas congress unite state joint...	0.9987
806	subject might become better silent speak diffi...	0.9994

As seen above, the compound score is the overall sentiment score for the sentence, normalized to range from -1 (most negative) to 1 (most positive). We can then use this score to label each sentence as negative (sentiment < -0.25), neutral (-0.25 < sentiment < 0.25), or positive (sentiment > 0.25). This choice is arbitrary and intended to achieve better class balance.

In [16]:
```
# Define a function to label each sentiment as positive, negative, or neutral.
def get_sentiment_label(compound_score):
    if compound_score > 0.25:
        return 'positive'
```

```
    elif compound_score < -0.25:
        return 'negative'
    else:
        return 'neutral'

df['sentiment_label'] = df['score'].apply(get_sentiment_label)
```

In [17]:
```
# Define a function to numerically label each sentiment as positive = 1, negati
def get_sentiment_class(compound_score):
    if compound_score > 0.25:
        return 1
    elif compound_score < -0.25:
        return -1
    else:
        return 0

df['sentiment'] = df['score'].apply(get_sentiment_class)
```

In [18]:
```
df.head()
```

Out[18]:

	transcripts	score	sentiment_label	sentiment
0	fellow citizen senate house representative amo...	0.2023	neutral	0
1	one hand summon country whose voice never hear...	0.5318	positive	1
2	hand magnitude difficulty trust voice country ...	0.0772	neutral	0
3	conflict emotion dare aver faithful study coll...	0.5994	positive	1
4	dare hope execute task much sway grateful reme...	0.9396	positive	1

Our data set is in good shape to move on to the next step as we now have our class labels. Before that, let's check for class imbalance.

In [19]:
```
df['sentiment_label'].value_counts()
```

Out[19]:
```
positive    67263
neutral     59653
negative    25941
Name: sentiment_label, dtype: int64
```

There are overwhelmingly many observations labeled as positive (44%) and neutral (39%), which can be a good thing in this context. It seems that the negative sentiment only makes up about 17% of the observations, but we still have a siazeable number of observations. Overall, there there are enough observations in each class and class imbalance is not a cause for concern.

Feature Extraction

We've cleaned our text and pre-labeled it. However, we don't have any features that we can feed into a machine learning algorithm. Thus, the next step is to extract some features from our text (that is, converting text data into a numerical representation) which can later be

used as inputs in a machine learning model to classify presidential speech according to sentiment.

There are a few techniques that can be used to achieve this.

Bag of Words (BoW)

This is a simple technique that involves counting the frequency of each word in a text and representing the text as a vector of word frequencies.

In [20]:
```
# Import necessary packages
import sklearn
from sklearn.feature_extraction.text import CountVectorizer

# Create CountVectorizer object
count = CountVectorizer()

# Fit the CountVectorizer to the preprocessed text data
X_bag = count.fit_transform(df['transcripts'])
```

In [21]:
```
print(X_bag.shape)
```

```
(152857, 28148)
```

Term Frequency-Inverse Document Frequency (TF-IDF)

This method of feature extraction takes into account the importance of words in a sentance relative to their frequency in the data set. Words that occur frequently in a sentence but rarely in the data set are given a higher weight. This helps to reduce the importance of common words like "the" and "and" and give more weight to important words.

In [22]:
```
# Import necessary packages
from sklearn.feature_extraction.text import TfidfTransformer

# Fit Countvectorizer to the preprocessed text data
counts = count.fit_transform(df['transcripts'])

# Create TfidfTransformer object and fit it to the precomputed term frequencies
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(counts)
```

In [23]:
```
print(X_tfidf.shape)
```

```
(152857, 28148)
```

Merging Features

We've extracted 2 key feature vectors that we will feed into our machine learning models. However, machine learning algorithms often accept only one input so we need to merge the 2 sparce matrices into 1 sparse matrix.

Sparse matrices are efficient data structures for working with large, high-dimensional datasets where most of the entries are zero. It is often recommended to use sparse matrices for large datasets with many features to save memory and speed up computations. Fortunately, the sci-kit learn implementations of the classification algorithms accept sparse matrices as an input.

In [24]:
```
# Import packages
from scipy.sparse import csr_matrix, hstack

# Concatenate the two sparse matrices horizontally
X = hstack([X_bag, X_tfidf])
```

In [25]:
```
print(X.shape)
```

```
(152857, 56296)
```

In [26]:
```
# Class (sentiment) labels
y = df.iloc[0:152857, 3].values
```

In [27]:
```
print(y.size)
```

```
152857
```

We are now ready to fit a classification model on our data with 152857 instances and 56296 features.

Model Training

Classification task:

We will use machine learning to train models to classify presidential speech by sentiment into one of 3 classes: positive, neutral, or negative.

Train and Test Data Split

We will perform a 70-30 split on our data set where 70% of observations will fall in the training set and the remaining 30% will be in the test set. The stratify command allows us to have the same proportion of class observations in both sets.

In [28]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =\
        train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

Logistic Regression

Parameter Optimization Using Grid Search

In [32]:
```
from sklearn.model_selection import GridSearchCV
import numpy as np
```

```
logit_parameters = {
    'C': [0.01, 0.1, 1, 5, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear'],
    'multi_class': ['ovr']
}
```

Fitting the Model

In [33]:
```
from sklearn.linear_model import LogisticRegression

# Set the random seed
np.random.seed(67)

# Fit the model
lr = GridSearchCV(LogisticRegression(), logit_parameters)
lr.fit(X_train, y_train)
lr_accuracy = lr.score(X_test, y_test)

# Print the best hyperparameters found by GridSearchCV
print("Best hyperparameters: ", lr.best_params_)

# Print the model accuracy
print("Classification accuracy: {:.2f}".format(lr_accuracy))
```

```
Best hyperparameters:  {'C': 1, 'multi_class': 'ovr', 'penalty': 'l1', 'solve
r': 'liblinear'}
Classification accuracy: 0.90
```

Using parameter optimization, we were able to construct a logistic regression model that has an accuracy rate of 0.90 -- that is, 90% of the observations were able to be classified accurately according to sentiment. This is quite promising as it provides a framework through which we can classify presidential speech by sentiment and identify patterns, biases, etc.

Limitations in computing power made it difficult to explore other more complex models in depth, and this was the best-perfoming model we were able to find.

Alternative Analysis

Finally, let's attempt to see whether the accuracy rate differs for presidential speeches from the first party era (1792 to 1824) and the sixth party era (1864 to present). This can allow us to have a sense on whether there was more polarity in different eras.

First Party Era

Prepare the Data Set

In [46]:
```
# Select only the text column
df1 = first_party_corpus[['transcripts']]

# Tokenize text -- split each row by sentence to create multiple rows
```

```
df1 = (
    pd.DataFrame(first_party_corpus["transcripts"].str.split(".").tolist(), inc
    .stack()
    .reset_index()
    .rename(columns={0: "transcripts"})
    .drop("level_0", axis=1)
    .drop("level_1", axis=1)
)

# Transform each sentence into lower case
df1["transcripts"] = df1["transcripts"].apply(lambda x: x.lower())

# Replace any non-word character with an empty string
df1["transcripts"] = df1['transcripts'].str.replace(r"[^\w\s]",'', regex = True

# Replace any number with an empty string
df1["transcripts"] = df1["transcripts"].str.replace('\d+', '', regex = True)

# Remove spaces at the beginning or end of the string
df1["transcripts"] = df1["transcripts"].apply(lambda x: x.strip())

# Remove the stop words
df1['transcripts'] = df1['transcripts'].apply(lambda x: ' '.join([word for word

# Lemmatize the sentences
df1['transcripts'] = df1['transcripts'].apply(lemmatize_words)

# Create a new 'sentiment' column
df1['score'] = df1['transcripts'].apply(get_sentiment_score)

df1['sentiment_label'] = df1['score'].apply(get_sentiment_label)
df1['sentiment'] = df1['score'].apply(get_sentiment_class)

# Bag of Words
X_bag1 = count.fit_transform(df1['transcripts'])

# TF-IDF
counts1 = count.fit_transform(df1['transcripts'])
X_tfidf1 = tfidf_transformer.fit_transform(counts1)

# Concatenate the two sparse matrices horizontally
X1 = hstack([X_bag1, X_tfidf1])

# Class (sentiment) labels
y1 = df1['sentiment'].values

# Train and test set split
X_train1, X_test1, y_train1, y_test1 =\
        train_test_split(X1, y1, test_size=0.3, random_state=1, stratify=y1)
```

Fit the Regression Model

In [47]:

```
# Fit the model
lr1 = GridSearchCV(LogisticRegression(), logit_parameters)
lr1.fit(X_train1, y_train1)
lr1_accuracy = lr1.score(X_test1, y_test1)

# Print the best hyperparameters found by GridSearchCV
print("Best hyperparameters: ", lr1.best_params_)
```

```
# Print the model accuracy
print("Classification accuracy: {:.2f}".format(lr1_accuracy))
```

```
Best hyperparameters:  {'C': 10, 'multi_class': 'ovr', 'penalty': 'l1', 'solve
r': 'liblinear'}
Classification accuracy: 0.73
```

The accuracy rate of this model is much lower than the overall model at 73%. More than a quarter of observations were misclassified by our best logistic regression model.

In [48]:
```
df1['sentiment_label'].value_counts()
```

Out[48]:
```
positive    2296
neutral     1332
negative     657
Name: sentiment_label, dtype: int64
```

There doesn't appear to be much class imbalance, so that is not the reason for the drastically lower accuracy rate. Perhaps vocubalary used during that time wasn't as indicative of sentiment and there was much lower polarity in speech, making it harder to accurately classify speech.

Sixth Party Era

Prepare the Data Set

In [49]:
```
# Select only the text column
df6 = sixth_party_corpus[['transcripts']]

# Tokenize text -- split each row by sentence to create multiple rows
df6 = (
    pd.DataFrame(sixth_party_corpus["transcripts"].str.split(".").tolist(), inc
    .stack()
    .reset_index()
    .rename(columns={0: "transcripts"})
    .drop("level_0", axis=1)
    .drop("level_1", axis=1)
)

# Transform each sentence into lower case
df6["transcripts"] = df6["transcripts"].apply(lambda x: x.lower())

# Replace any non-word character with an empty string
df6["transcripts"] = df6['transcripts'].str.replace(r"[^\w\s]",'', regex = True

# Replace any number with an empty string
df6["transcripts"] = df6["transcripts"].str.replace('\d+', '', regex = True)

# Remove spaces at the beginning or end of the string
df6["transcripts"] = df6["transcripts"].apply(lambda x: x.strip())

# Remove the stop words
df6['transcripts'] = df6['transcripts'].apply(lambda x: ' '.join([word for word

# Lemmatize the sentences
df6['transcripts'] = df6['transcripts'].apply(lemmatize_words)
```

```
# Create a new 'sentiment' column
df6['score'] = df6['transcripts'].apply(get_sentiment_score)

df6['sentiment_label'] = df6['score'].apply(get_sentiment_label)
df6['sentiment'] = df6['score'].apply(get_sentiment_class)

# Bag of Words
X_bag6 = count.fit_transform(df6['transcripts'])

# TF-IDF
counts6 = count.fit_transform(df6['transcripts'])
X_tfidf6 = tfidf_transformer.fit_transform(counts6)

# Concatenate the two sparse matrices horizontally
X6 = hstack([X_bag6, X_tfidf6])

# Class (sentiment) labels
y6 = df6['sentiment'].values

# Train and test set split
X_train6, X_test6, y_train6, y_test6 =\
        train_test_split(X6, y6, test_size=0.3, random_state=1, stratify=y6)
```

Fit the Regression Model

In [50]:
```
# Fit the model
lr6  = GridSearchCV(LogisticRegression(), logit_parameters)
lr6.fit(X_train6, y_train6)
lr6_accuracy = lr6.score(X_test6, y_test6)

# Print the best hyperparameters found by GridSearchCV
print("Best hyperparameters: ", lr6.best_params_)

# Print the model accuracy
print("Classification accuracy: {:.2f}".format(lr6_accuracy))
```

```
Best hyperparameters:  {'C': 1, 'multi_class': 'ovr', 'penalty': 'l1', 'solve
r': 'liblinear'}
Classification accuracy: 0.89
```

This model performed in line with our overall model with an accuracy rate of 89%, which might be indicative of higher polarity in recent times than in the past. That is, it is much easier to classify text according to sentiment, perhaps due to strength in opinion and messaging, and that is promising for the applications of this work.

In [51]:
```
df6['sentiment_label'].value_counts()
```

Out[51]:
```
neutral     29618
positive    29361
negative    11891
Name: sentiment_label, dtype: int64
```

Observe that we also have more data for the sixth party system as compared to the first party system, which may have resulted in a better performing model. In fact, almost half of our entire data set is composed of sixth party system speeches, so that must have carried a lot of weight in our comprehensive model.

Something that stands out, however, is the fact that the sixth party system has a majority of neutral observations, which stands out in comparison to the entire data set and the first party system. This indicates that a lot of speeches carried both negative and positive sentiment and the overall sentiment score ended up being neutral.

Naive Bayes Classifier

Finally, let's see whether a Naive Bayes classifier, which is commonly used for sentiment analysis, will produce a model with higher accuracy.

In [57]:
```
from sklearn.naive_bayes import MultinomialNB

# Fit the Model
model = MultinomialNB()
model.fit(X_train, y_train)
model_accuracy = model.score(X_test, y_test)

# Print the model accuracy
print("Classification accuracy: {:.2f}".format(model_accuracy))
```

```
Classification accuracy: 0.73
```

The accuracy rate is quite low and may be due to model complexity.

Logistic regression performs quite well for our purposes.

Results and Visualizations

We were able to build a model with 90% accuracy, which is very promising since only 10% of observations were misclassified. It's important to note that the model is a simple logistic regression model, so given more computing power and time, there was potential to build even more powerful models that can allow us to classify presidential speeches more accurately.

We've been able to build a classification model, but let's complete this holistic analysis by visualizing what our data looks like in the first place.

In [65]:
```
#pip install wordcloud
```

In [62]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Concatenate all the sentences into a single string
text = ' '.join(df['transcripts'])

# Create a wordcloud object
wordcloud = WordCloud(width=800, height=400, background_color='white').generate

# Display the wordcloud
plt.figure(figsize=(10, 5))
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Overall, we see a lot of formal diplomatic language such as "state", "government", "unite", "people", "nation", and "country".

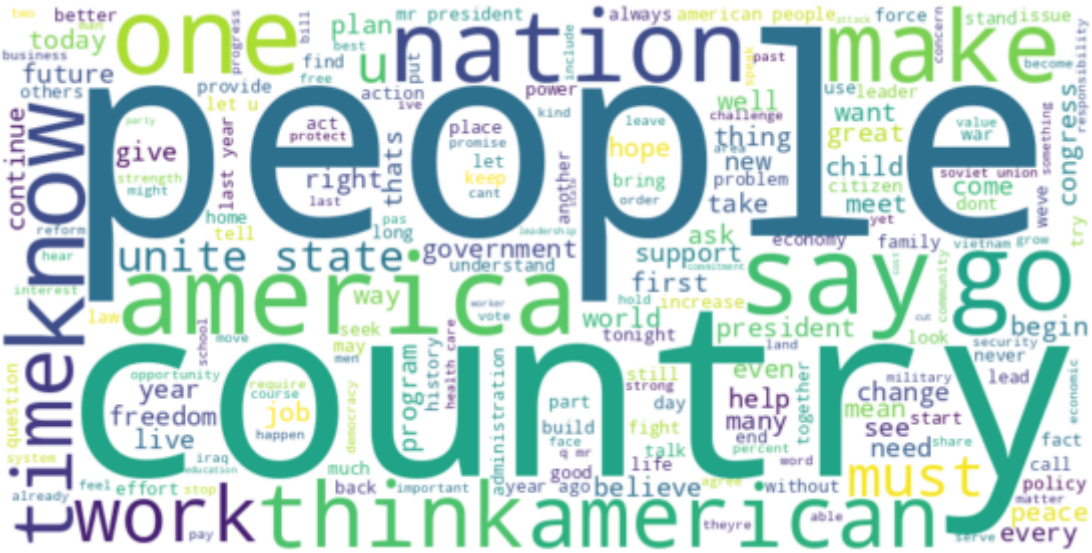
Let's see whether the language differs when we compare the first party and sixth part system.

First Party System

In [63]:
```
# Concatenate all the sentences into a single string
text1 = ' '.join(df1['transcripts'])

# Create a wordcloud object
wordcloud1 = WordCloud(width=800, height=400, background_color='white').generat

# Display the wordcloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud1, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Sixth Party System

In [64]:
```
# Concatenate all the sentences into a single string
text6 = ' '.join(df6['transcripts'])

# Create a wordcloud object
wordcloud6 = WordCloud(width=800, height=400, background_color='white').generat

# Display the wordcloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud6, interpolation='bilinear')
plt.axis('off')
plt.show()
```



There is quite a difference when we compare the two different eras. Something noticeable is the emphasis on "America" and "American" in the sixth party system that is not apparent in the first party system. We also see that "people" and "country" are overwhelmingly used in comparison to the earlier system. A lot of similar words come up, however, but it's interesting to observe the shift in the presidential speech language.

Next, let's take a deeper look at the words our lexicon is classifying as positive and negative.

Positive

In [74]:
```
positive_df = df[df['sentiment_label'] == 'positive']

# Concatenate all the sentences into a single string
textp = ' '.join(positive_df['transcripts'])

# Create a wordcloud object
wordcloudp = WordCloud(width=800, height=400, background_color='white').generat

# Display the wordcloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloudp, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Negative

In [75]:
```
negative_df = df[df['sentiment_label'] == 'negative']

# Concatenate all the sentences into a single string
textn = ' '.join(negative_df['transcripts'])

# Create a wordcloud object
wordcloudn = WordCloud(width=800, height=400, background_color='white').generat

# Display the wordcloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloudn, interpolation='bilinear')
plt.axis('off')
plt.show()
```

We see a lot of the same words appear in both positive and negative speech, which the exception of "war", which is regarded as negative. Besides this, words like "government", "unite", "state", "people, and "country" appear predominantly in political speech despite the sentiment.

What this really shows us is that the words supporting these key themes is what plays a key role in determining sentiment. We see in smaller font words like "must", "enemy", "attack", and "wrong" in the negative speech as opposed to words like "may", "support", "believe", "great", "secure", and "respect" in the positive speech. This gives us greater insight into why classification of political speech can be difficult task. Some themes pervade regardless of era, sentiment, nation, etc, but it is the finer font that truly allows us to determine the polarity and sentiment of the speech. More training data may be essential to allow our algorithms to read in between the fine print.

Conclusion and Implications

In conclusion, we have fulfilled our goal and built a model that can classify presidential speech by sentiment accurately 90% of the time as well as began to identify some differences across political eras and sentiment classes. There is certainly some shift in language as well as some indication that it is easier to classify more recent data, perhaps due to an abundance of it.

This work has potential for many implications. It can be invaluable for understanding the state of political speech in the United States on a national scale. This can provide insights into various aspects of politics and have direct impact by improving political communication and allowing politicians to tailor their messaging to be more effective and engaging with their target audience, identifying bias, propaganda, or misinformation that may be present, and improving political campaign strategies and policy-making.

Related Work

Gavin Abercrombie and Riza Batista-Navarro. 2020. ParlVote: A Corpus for Sentiment Analysis of Political Debates. In Proceedings of the Twelfth Language Resources and Evaluation Conference, pages 5073–5078, Marseille, France. European Language Resources Association.

This work looks at sentiment analysis of debate speeches, specifically from the UK parliament, and found that in many scenarios, a linear classifier trained on a bag-of-words text representation achieves the best results.

John Paul P. Miranda & Rex P. Bringula | John Kwame Boateng (Reviewing editor) (2021) Exploring Philippine Presidents' speeches: A sentiment analysis and topic modeling approach, Cogent Social Sciences, 7:1, DOI: 10.1080/23311886.2021.1932030

This study analyzed the annual obligatory and traditional speeches, referred to as State of the Nation Address (SONA), of the 13 past Philippine presidents. The study determined the sentiments, as well as the emergent topics, expressed in these materials. It is found that these SONAs generally expressed positive sentiments while the lowest negative sentiment, on the other hand, was during the martial law period in 1974.

Elena Rudkowsky, Martin Haselmayer, Matthias Wastian, Marcelo Jenny, Štefan Emrich & Michael Sedlmair (2018) More than Bags of Words: Sentiment Analysis with Word Embeddings, Communication Methods and Measures, 12:2-3, 140-157, DOI: 10.1080/19312458.2018.1455817

Moving beyond the dominant bag-of-words approach in sentiment analysis, this study introduces an alternative procedure based on distributed word embeddings for sentiment analysis on Austrian parliamentary speeches.

Suggestions for Future Work

There were limitations to this study with regard to time and computing power. Exploring more complex models using ensemble methods like random forest or classifiers that are effective in high dimensional spaces such as support vector machines (SVM) may produce a model with higher accuracy. These models were unable to run in reasonable time or without restarting the kernel on my device, but it may be worth exploring if time and resources allow. Nevertheless, logistic regression performs quite well.

Additionally, I used the VADER lexicon to pre-classify my data set. There may be other methods of sentiment classification that may yield better results.

Further study can use this framework and model to look into party data, state data, regional data, or even global data to identify similarities and differences in political messaging with regard to sentiment.

Ultimately, a model like this can be employed to aid presidents in crafting their speeches and perhaps this work can be extended to other forms and mediums of political messaging such as that through social media or for the purposes of campaign.