# Project Tutorial: Topic Modeling the State of the Union

Using SpaCy, CorEx, and KMeans Clustering to understand presidential addresses through history, and analyze the 2022 SOTU.

Our project for this module was to bring our newly-acquired NLP skills to bear on a dataset, analyze the text, and generate insights. Because the 2022 State of the Union was delivered right at the beginning of the module, I thought it would be interesting to analyze State of the Union speeches through history to see if there were common topics they all covered and if we could use the topic distributions to group the addresses into clusters that made sense thematically.

A quick note about NLP terminology

We're going to be throwing around some strange words so I'd like to provide some definitions up front:

- Token: A token is a word, or group of words, that has been cleaned and processed as part of the NLP pipeline. For example, with NLTK''s Word Tokenizer 'New' and 'York' would be two tokens, but if we applied named entity recognition to 'New York', we would get one token with a completely different meaning.
- Document: In NLP, A Document is a distinct collection of tokens within the broader dataset. Documents can be of varying size — In some use cases, a tweet might be a document. In this project, each State of the Union speech is a document. Generally, they are the distinct linguistic entities being analyzed.
- Corpus: A Corpus is a collection of documents, and can be thought of as the overall dataset in an NLP project. In this project, all of the State of the Union addresses collectively is our corpus.

**Why Analyze the State of the Union Address?**

In today's world, the State of the Union is a major political spectacle. It includes pomp, circumstance, and wall-to-wall coverage on all the major news networks. Presidents use it as an opportunity to speak directly to a large portion of the American public, and often tout the accomplishments of their administration, unveil new policy initiatives, signal future priorities, and test party messaging for upcoming election cycles.

Beyond the things a president chooses to highlight in their speech, State of the Union addresses are also reflective of the zeitgeist in which they happen. When George Washington delivered the first address in 1790, he congratulated Congress on the recent accession of North Carolina to the Constitution right at the beginning. In Joe Biden's 2022 State of the Union, he denounced Vladimir Putin's invasion of Ukraine that had happened six days prior.

Because these texts include references to important events in the recent past and a president's priorities for the near future. I thought it would be worthwhile to analyze all of them going back to 1790 to identify the key topics discussed in each, and see if there are any commonalities.

**Dataset**

Luckily for me, a Kaggle dataset with every State of the Union speech from 1790–2019 was readily available. All I had to do was manually add in the 2022 speech after it was given, and I was ready to get to work. The updated CSV file is contained in my GitHub repository.

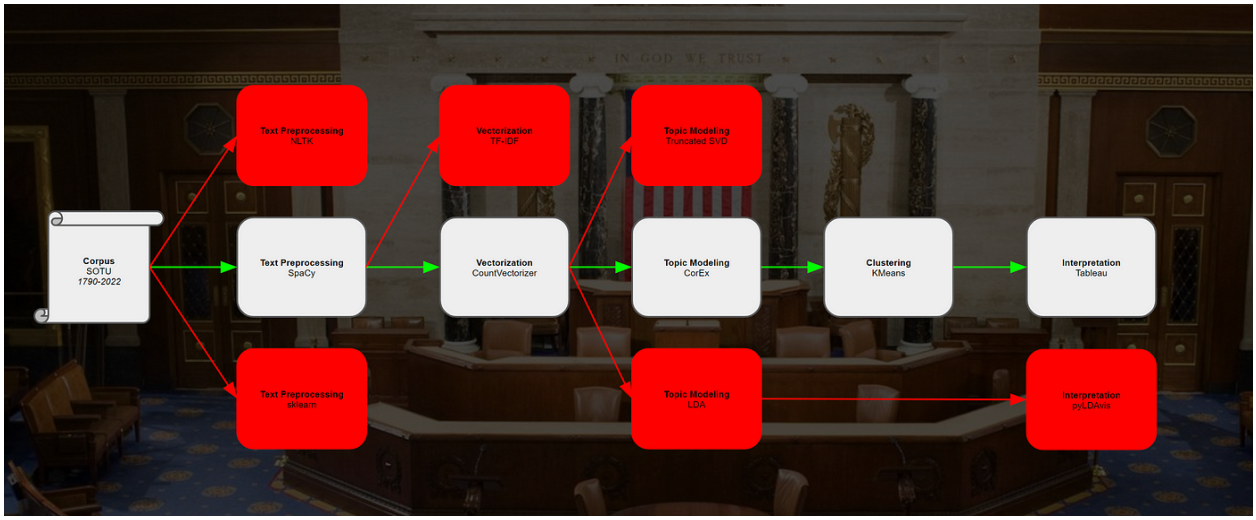
**Goals and Success Metrics**

One facet of working in the NLP and Unsupervised Learning fields is that there are no tried and true evaluation metrics for results. While we can use packages such as pyLDAvis, or proximal metrics like KMeans' inertia and WCSS to determine and fine-tune how coherent our groupings are, NLP/Unsupervised Learning is just as much art as it is science. Results need to be manually gut-checked (i.e. Do these clustering's make sense?), and oftentimes unsupervised learning is applied as a step in a supervised learning pipeline (e.g. sentiment score where tweets are rated positive or negative). All this is to say that defining success with an unsupervised project is ambiguous, and while that may present a challenge in some respects, it also allows for a degree of latitude that can make things fun!

For my project, I decided that I wanted to find the top 10 topics across all of the addresses since 1790. Then, I wanted to find the distribution of topics for each address (e.g. 20% economy, 40% military, etc.). Finally, I clustered the speeches according to their topic distributions and checked the results to see if speeches that covered similar topics in similar proportions ended up in the same cluster.

**The Unsupervised/NLP Workflow**

Given unsupervised learning's ambiguous nature, a lot of trial and error went into determining the final solution path for this project.




Workflow for this project. Image by Author.

With the aforementioned lack of evaluation metrics for unsupervised learning, determining what the final solution path was going to be required me to manually check the results of the different methods and packages mapped above, and choose what I thought was best based on my interpretation (which was driven in part by some small amount of domain knowledge I have from a previous life).

I think it's important to note that clean and linear narrative blog posts like these present projects are not true to reality. Data science projects in general, and NLP/Unsupervised learning in particular, are messy processes that require trial and error, numerous iterations, and a fair degree of flexibility.

Further, each stage of the solution path required me to make choices about how I was working with the data, and those choices had significant impacts on the results. The text preprocessing methods, the Vectorizer used, the topic model used, all of these have impacts on the final result, and no result is objectively more valid than another.

**Code, Models, and Results**

Packages

Here's a quick rundown of the packages used in this project:

```
import pandas as pd
import numpy as np
import string
import re
import spacy
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import TruncatedSVD, LatentDirichletAllocation
```

```
from corextopic import corextopic as ct
from corextopic import vis_topic as vt
from matplotlib import pyplot as plt
import pyLDAvis
import pyLDAvis.sklearn
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from sklearn.preprocessing import minmax_scale
import seaborn as sns
%matplotlib inline
```

**Text Preprocessing**

Before we can work with our texts, we need to pre-process them to be more easily understood by a computer. Pre-processing includes the removal of punctuation and capitalizations, stemming/lemmatizing of individual tokens, part of speech tagging, named entity recognition, and stop word removal, to name a few.

Manually pre-processing text in an NLP project is a huge pain. Packages like NLTK and sklearn offer tools for pre-processing, but they have to be used in a specific order to ensure the best results. Luckily, the SpaCy package exists, and it offers processing pipelines with all of the requisite steps included out of the box. SpaCy makes life easy, and I highly recommend it:

```
df = pd.read_csv('sotu_texts.csv')
nlp = spacy.load("en_core_web_sm")
docs = nlp.pipe(df.Text)
docs_clean = [[w.lemma_.lower() for w in doc if (not w.is_stop and not w.is_punct and not w.like_num)] for doc in docs]
df['docs_clean'] = docs_clean
```

**Stop Words**

In NLP, Stop Words are words that hold little to no meaning in our document. ‘The’, ‘I’, ‘and’, all appear frequently, but don’t tell us anything about the content. Because of that, it is usually best practice to remove stop words as part of the Text Preprocessing phase. sklearn, NLTK, and SpaCy all contain their own list of stop words. Often, however, our use case necessitates the removal of additional stop words specific to our domain.

In this project, words like ‘American’, ‘country’, ‘national’, and ‘people’ didn’t convey much meaning, so we added them as custom stop words to be removed:

```
stop_words = text.ENGLISH_STOP_WORDS.union(['American', 'people', etc...])
```

**Vectorization**

Once our raw text has been processed into clean documents, we need to reframe our documents into a document-term matrix. A document-term matrix takes the documents we've created and represents them as rows in a table, with the columns of that table being a value representative of the words present in our corpus.
With sklearn's CountVectorizer, that value is a simple count, while with Term Frequency-Inverse Document Frequency Vectorizer the value represents the importance of each word by weighting the term's frequency within a document relative to the overall corpus. More weight is placed on words that appear infrequently across the corpus, but frequently within a single document, while words that appear across all documents frequently receive less weight. As with all things NLP, Vectorization has an impact on the final results, and so the method that works best for your use case will depend on your data.
Beyond the different methods of vectorization, sklearn's implementation also has hyperparameters that can be tuned to influence the results. Min and max df (document frequency) set a cutoff point for the maximum or a minimum number of documents a token appears in, and ngram_range=(1,3) analyzes every 1–3 word combination as a potential token:

```
count_vec = CountVectorizer(stop_words=stop_words, ngram_range=(1,3), min_df=2,
max_df=219)
count_vec_X = count_vec.fit_transform(docs_list_clean)
tfidf_vec = TfidfVectorizer(stop_words=stop_words, ngram_range=(1,3), min_df=2,
max_df=219)
tfidf_X = tfidf_vec.fit_transform(docs_list_clean)
```

For this project, I chose to use CountVectorizer after comparing the results with Tf_idf.

**Topic Modeling**

With our document-term matrix, we can finally do some topic modeling! There are numerous topic models, including Latent Semantic Analysis, and Latent Dirichlet Allocation. Topic Models are a form of dimensionality reduction, not dissimilar from Principal Component Analysis. We can use them to break our documents into 'topics', which can be thought of as a probability distribution of words. Documents are not exclusively one topic or the other, but rather are a blend of a number of topics, with each topic having words most closely associated with it.
Note: Topic Models can make some…unfortunate associations. I do not endorse these results as valid or coherent.

```
LDA_topics = LatentDirichletAllocation(n_components=10)
count_vec_X = count_vec.fit_transform(docs_list_clean)
def display_topics(model, feature_names, no_top_words, topic_names=None):
for ix, topic in enumerate(model.components_):
if not topic_names or not topic_names[ix]:
print("\nTopic ", ix)
else:
```

```
print("\nTopic: '",topic_names[ix],"'")
print(", ".join([feature_names[i]
for i in topic.argsort()[:-no_top_words - 1:-1]]))

display_topics(LDA_topics, count_vec.get_feature_names_out(), 10)
```

LDA Selected Topics:

- act war, dictator, armistice, capture british, command captain, french decree, repeal order, population resource, impressive, world freedom
- saddam hussein, hussein, saddam, inspector, disarm, aids, weapon, north korean, biological, hide
- weapon development, nuclear weapon development, send propose, command control, iran continue, international food, generate energy, alliance world, economic stagnation, major research
- terrorist, iraq, mexico, iraqi, bank, al, enemy, terror, texas, war

As we can see, there are some interesting results. It seems as though Saddam Hussein, inspectors, biological, weapon, and North Korean are lumped together, and if you're old enough to remember the Bush administration's 'Axis of Evil', that makes sense.

The model also saw a connection between the words terrorist, Iraq, Iraqi, Mexico, Texas, and war. As I said, some topics have unfortunate implications, but there is a utility to be had here — hypothesizing that presidents speak about the Mexico/Texas border in the same terms they do war and terrorists is something worthy of further analysis, but not the ultimate goal of this project.

Because the results with LSA/LDA weren't great, I decided to try a different topic model. Correlation Explanation is a bit better at handling longer documents, and also has the ability to use anchor words for semi-supervised learning. Without anchor words, the results still weren't great:

CorEx Unsupervised Results:

- custom, make, rigidly, postage, postmaster, benefit, favorable consideration, expend, postmaster general, aggregate
- clause, convinced, organization, value, product, outlet, investigation, topic, absence, condemn
- opinion, vest, network, coalition, Iran, regard, practicable, Iraq, arrangement, modernize

So I used some anchor words like 'labor' 'economy' 'law' 'war' 'research' and so on to define my 10 categories using semi-supervised learning. I was interested in seeing what the model found when I gave it a warm start:

CoreEx Semi-Supervised Results:

- program, job, economic, today, budget, help, goal, ahead, basic, worker
- crop, farmer, agriculture, farm, method, production, business, department commerce, type, interstate commerce

- cent, tariff, fiscal, expenditure, currency, treasury, end june, suggestion, postmaster, fiscal end
- school, student, education, college, university, math, street, teacher, medicare, tax credit

These topics were more coherent and distinct when compared to the unsupervised results, so I decided to cluster the speeches based on the distributions of the Top 10 topics they possessed.
Top 10 Topics:

- Labor
- Economy
- Federal Government
- Finances/Budget
- Social Issues
- Foreign Policy
- Science and Technology
- Law/Jurisprudence
- Education
- Infrastructure

Last, I joined the top topics from our CorEx model into our original dataframe, for later use:

```
predictions = pd.DataFrame(topic_model.log_p_y_given_x)
predictions['top_topics']=predictions.idxmax(axis=1)
df['top_topic'] = predictions.top_topics
```

Interestingly, nearly every speech since 1943 fell into the Economy/Labor category, including Joe Biden's 2022 Speech.

**Clustering**

To cluster the speeches based on their topic distributions, I took the output of our CorEx model, applied a min/max scaling method, and fit a KMeans model:

```
predictions_scaled =
minmax_scale(np.array(predictions.drop(columns=['top_topics']).values.T)).T
preds_df = pd.DataFrame(predictions_scaled, columns=['Economy/Labor',
'Economy/Manufacturing', 'Government/Federalism/States', 'Finances/Budget', 'Social Issues',
'Foreign Policy', 'Science and Technology', 'Crime/Law/Jurisprudence', 'Education',
'Infrastructure'])
mtx = preds_df.values
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(mtx)
df['Cluster'] = kmeans.labels_
```

To gut-check the results, I can check the top the documents for a given cluster:

```
top_3_docs = pairwise_distances([[num[0] for num in df_clusters.iloc[1].values.reshape(-1,
1).tolist()]], mtx,metric='euclidean').argsort()[0, :3]
df.iloc[top_3_docs[0:3]]
```

This returned Andrew Jackson's 1834, 1831, and 1836 State of the Union addresses, the content of which was somewhat varied. It was interesting, however, that most clusters' top 3 documents were from the same president, or a very similar time period. I would be interested to know how much of the clustering happened along thematic lines like I'd intended, as opposed to pure linguistic similarity as seems to be the case here (i.e. the speeches were clustered according to which ones had the most similar word choice, and therefore by distinct linguistic epochs in American English).
In addition to the majority of the speeches after 1943 having a top topic of Labor, they were also clustered almost exclusively in Cluster 2. Again, this may be a matter of linguistic similarity, and would need further analysis.

**(Bonus) Per-Speech topic models**

I didn't find the results of my topic models or clusters particularly satisfying, so I decided that I would get better results with topics on a per-speech basis. I didn't have a lot of time to explore here but I had some very interesting initial results:

```
for i in df.docs_clean:
x = count_vec.fit_transform(i)
y = LDA_topics.fit_transform(x)
display_topics(LDA_topics, count_vec.get_feature_names_out(), 5)
```

Biden 2022 Topics:

- work, pass, child, care, justice, announce, come, ukraine, company, pandemic, continue, ago, fund, end, leave, grow, liberty, rate, send, choose
- need, price, law, protect, life, officer, strong, school, community, thing, ready, ally, forward, union, federal, tell, assistance, manufacturing, power, order
- job, build, home, economy, pay, cut, putin, act, look, free, help, long, police, sure, worker, moment, freedom, disease, right, day
- family, cost, fight, tax, good, business, history, economic, violence, high, lower, russian, treatment, deficit, happen, low, million, force, inflation, department
- plan, support, russia, stop, cancer, provide, health, ukrainian, test, president, month, create, meet, veteran, vaccine, want, mask, investment, corporation, wage

Trump 2019 Topics:

- border, woman, life, wall, family, ago, legislation, city, brave, class, agenda, century, confront, right, step, girl, elvin, matthew, nearly, barrier
- work, come, ask, child, war, alice, citizen, southern, choose, promise, price, create, economic, judah, save, defend, economy, freedom, god, troop
- join, trade, end, herman, soldier, prison, protect, begin, reduce, young, today, joshua, agent, grace, good, unfair, safe, moment, trafficker, low
- pass, administration, fight, illegal, law, reach, build, party, thank, progress, act, middle, face, political, immigration, pay, china, nuclear, agreement, cancer

- decade, job, drug, day, include, month, secure, home, historic, president, place, hero, bring, beautiful, dream, community, fellow, death, patient, story

Conclusions

This was a really interesting project! And though the results were a bit ambiguous, I think there were some interesting takeaways that could be grounds for future work:

1. Topic models do not perform well on long form text like speeches. Even the most advanced deep learning models like BERT and GPT-3 struggle with documents beyond a paragraph in length. I had much more success with the per-speech topic models than I did the topic models taking all of the text from every speech.
2. Speeches were potentially clustered by linguistic similarity, rather than thematic. This suggests that we could potentially partition speeches by epoch for better analysis, and could also help explain why every speech post-1943 was clustered into one group — because they sounded the most similar when compared with earlier addresses.
3. There are still some use cases for which a data-driven approach may not be best. While topic models helped us distill some of the core ideas addressed in a given speech, we were unable to make any definitive conclusions about thematic groupings. Perhaps we could do so with more advanced models, or a different text preprocessing approach, but talking to a political historian or other subject matter expert to unpack the layers of meaning present in a 1000+ word speech may be a more productive use of one's time in terms of understanding.

I enjoyed this project, because it taught me a valuable lesson about the limitations of data science and machine learning. With a use case like Natural Language Processing, machines can do some really amazing things, but still have a long way to go to catch up to our own language comprehension skills.

As I mentioned in the beginning, unsupervised learning is just as much art as it is science and in that respect, I enjoyed the process of trying to dial in coherent topics using some domain expertise and different models/preparation methods, and the flexibility in approach the lack of hard evaluation metrics afforded me. I hope someone finds this project insightful, interesting, or educational, and feel free to drop a line with any questions!