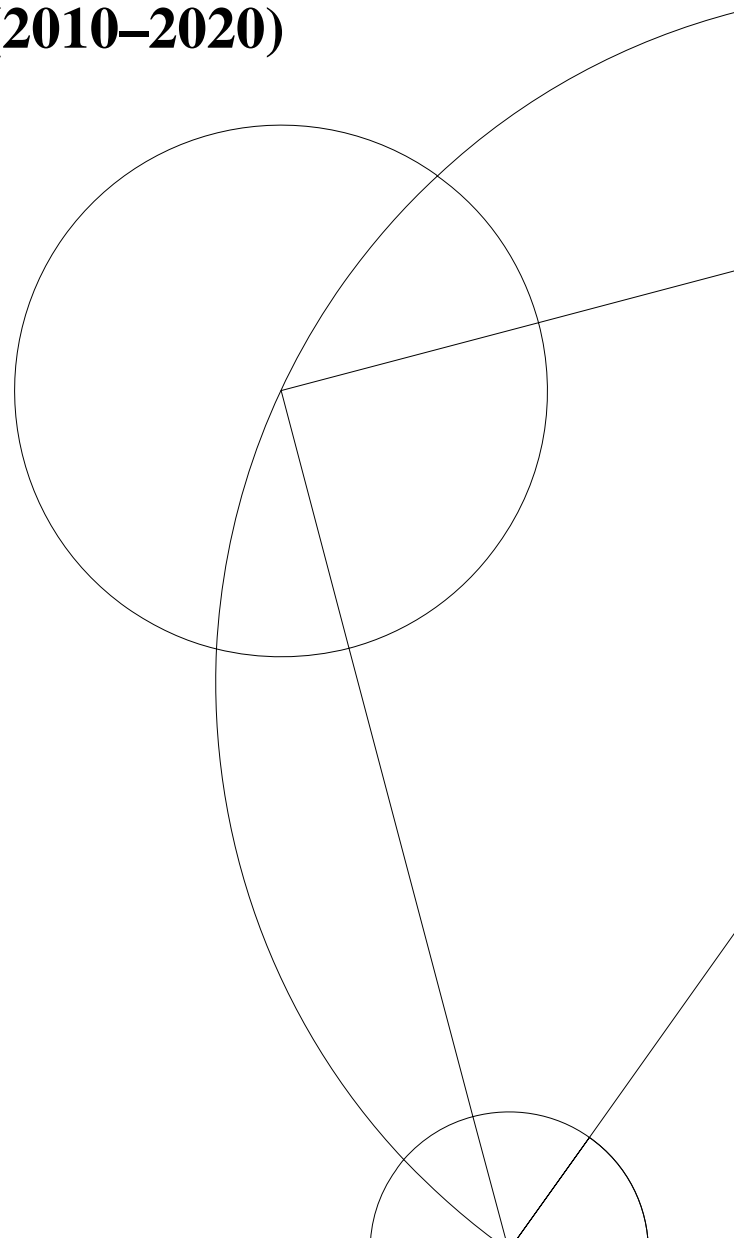
KØBENHAVNS UNIVERSITET

ASDS II Course

Asteriou Stavroula

Analyzing Government Rhetoric in Greek Parliamentary Proceedings (2010–2020)



Character Count: 22,800 (9,5 pages)

Spring 2025

Contents

1 Introduction

This assignment for the examination of ASDS II was designed to meet the course requirements by engaging with a dataset that is both rich in textual detail and highly relevant to computational social science. The project aims to apply a wide range of methods introduced in the course to address distinct and relevant research questions about political rhetoric. To deepen my understanding of the course material, particularly the challenging of preprocessing and analysing non-English text, I chose to work with data in Greek, my native language. Greek poses additional challenges due to its unique alphabet, inflectional morphology and complex syntactic structures, while text computational greek text analysis remains relatively underexplored compared to english and other widely used languages.

The dataset under study consists of parliamentary proceedings from the Hellenic Parliament from 2010 to 2020. These proceedings are publicly available on the Hellenic Parliament's website [1], but for purposes of this project I relied on a enriched version compiled by Dritsa (2018) on behalf of iMEdD [2]. The original corpus contained 1.28 million speeches (1989-2020). After filtering to the years 2010–2020, the corpus used here consists of 536,446 documents, accompanied by metadata on speaker name,political party, government, roles of speakers, sitting date, parliament session, member name, gender and region.

The aim of this project is to reconcile different methods covered on course and apply them to distinct aspects of political speech and rhetoric style, focusing especially on the language of government officials, with inspiration from studies such as Hjorth (2024) and Dritsa (2018) . This leads to the three following research questions:

- RQ1: How do government and opposition parties differ in their emotional rhetoric during political crises?
- RQ2: Do government use more complex language compared to other MPs?
- RQ3: Has the government's rhetorical style shifted over time?

[1] https://www.hellenicparliament.gr/Koinovouleftikes-Epitropes/Synedriaseis

[2] Incubator for Media Education and Development, data accessed by: https://lab.imedd.org/en/parliamentary-minutes-from-1989-until-today

2 Theory

Rhetorical style is a central component of political communication reflecting political roles, policy content, and of course how messages are delivered to citizens, their simplicity, complexity and emotional tone, making it an important dimension of representation (Widmann, 2021). Governments in particular face unique rhetorical constrains, they must justify policies, maintain legitimacy, and often communicate under pressure of crises(Pancer et al., 1992; Widmann, 2021; Hjorth, 2024).

2.1 Emotional rhetoric

Emotions are central to political communication because they shape how citizens process information and evaluate politics (Widmann, 2021). For instance, populist parties in particular emphasize negative emotions such as anger,fear, disgust, and sadness to frame the political world in antagonistic terms, while non-populist parties more often employ positive emotions to highlight achievements and system stability. In parliamentary settings anger and disgust attribute blame, fear shows risks and hope or enthusiasm signal opportunity (Widmann, 2021). These dynamics underpin RQ1, which compares how government and opposition deploy emotional rhetoric during crises.

2.2 Linguistic complexity

Readability and rhetorical complexity are also key dimensions of parliamentary speech. Earlier studies show that government officials speak with greater complexity because they must justify policies and defend unpopular measures, while opposition MPs are freer to use simplified, populist rhetoric (Pancer et al., 1992; Hjorth, 2024). Hjorth (2024) formalizes this as the *rhetorical cost of governing*, noting that "legislators generally seek to satisfy voter demands for maximal rhetorical simplicity, the functional demands of holding government office compel government legislators to speak with less simplicity" (p.2). This literature supports the RQ2 on whether government officials in Greece use more complex language than other MPs.

2.3 Rhetorical change over time.

Finally, rhetorical style adapts to context, particularly in periods of crisis or partisan turnover (Widmann, 2021). Benoit et al. (2019) highlights broader concerns about declining rhetorical sophistication in political communication, while recent advances in distributional semantics enable systematic measurement of senmatic drift. Rodriguez and Spirling (2022) demonstrate that pretrained embeddings can capture even subtle shifts in political language over time. These motivated RQ3, which explores whether rhetorical style of Greek governments' speeches shifted across the decade.

3 Methods

3.1 Preprocessing for Greek Text

As Denny and Spirling (2018) emphasize, preprocessing choices are not neutral: different pipelines can alter conclusions by shaping how text is converted to numerical representations. Researchers face a trade-off between simplifying the data and retaining sufficient linguistic information. Also, Hovy (2021) [sect.2] and Hovy (2022)[sect. 3-3.1, 5.3] highlights the distinction between discrete representations, where vector position correspond to a word and distributed representations, where positions no longer correspond to specific words but capture latent semantic features. In this project, the first two analyses (RQ1 and RQ2) rely on discrete representations, while RQ3 employs distributed representations through contextual embeddings.

Greek text introduces additional challenges. The language has evolved over three millennia with a unique maintained alphabet, high inflection, and flexible word order, complicating preprocessing tasks. Moreover, Greek is under-represented in most multilingual corpora which limits general-purpose NLP models. GR-NLP-Toolkit (Loukas et al., 2024) provides the most advanced open-source functionality for Greek but technical incompatibilities with python and library versions in my environment required me to rely on spaCy. While spaCy supports lemmatization and tokenization for Greek, its reliance on static FastText embeddings rather than pre-trained transformers restricts

accuracy (Loukas et al., 2024).

General preprocessing involved removing diacritics, formal expression and honorifics to reduce noise, as well as extra punctuation, white space, and numbers were stripped. For lexicon-based sentiment detection, speeches were tokenized and lemmatized while stopwords were retained, since several appear in the lexicon produced by Dritsa (2018). Also, due to limited time and resources, sentence-level processing as in Dritsa (2018) original approach could not be replicated. For SMOG Index calculation, punctuation was kept to preserve sentence boundaries, digits were normalized and tokenization was used to identify polysyllabic words and sentence counts. For semantic drift with GreekBERT, the minimally cleaned text was used directly, as the pretrained model benefits from retaining morphological variation and linguistic signals.

3.2 Sentiment Detection (Lexicon-based)

To measure emotional rhetoric, I implemented a lexicon-based method following Dritsa (2018); Hovy (2022) [sect.5.2,5.2.4], who developed a manually annotated Greek emotion lexicon assigning scores across six basic emotions: anger, disgust, fear, happiness, sadness, and surprise. Each term in the lexicon is associated with a six-dimensional sentiment vector. While Dritsa (2018) originally applied a fuzzy-matching approach using Jaro-Winkler distance combined with manual validation of frequent terms, I adapted the procedure for feasibility. Specifically speeches were lemmatized and tokens were matched to lexicon entries through exact matches only, rather than approximate similarity scores. This simplification was necessary due to time and resource constraints, as the original approach required labor-intensive manual curation to resolve false positive in specialized parliamentary vocabulary. Although distributed approaches like Greek-BERT sentiment classification Hovy (2022)[sect. 3-3.1], could provide richer embeddings, I opted for the lexicon based approach to remain consistent with the exam requirement to demonstrate a variety of methods.

For scoring, each token matched to the lexicon contributed its six-dimensional emotion vector, and scores were aggregated at the document level using the root mean square methods. This produced a single sentiment profile across six emotions for each parlia-

mentary speech. A limitation of this approach is that lexicons are static and context-independent, so they cannot capture irony, shifting word meaning, or influence of surrounding words on emotional tone. Moreover, by relying on exact matches, the analysis risks are underestimating emotion when lexical variation (inflected forms not present in the lexicon) leads to missed matches (Dritsa, 2018; Proksch et al., 2019; Denny and Spirling, 2018).These limitations imply that the validity of my results may be somewhat reduced compared to Dritsa (2018)'s original implementation, which combined fuzzy matching with manual validation to better handle lexical variation and domain-specific vocabulary. My simplified approach prioritized scalability and exam requirements over fine-grained accuracy, making the results suitable for comparative analysis but potentially less precise in absolute measurement.

3.3 Readability & Complexity (SMOG Index)

To asses the linguistic complexity of parliamentary speeches, I applied the SMOG index, a standard readability metric originally designed for English texts. In this conventional form, the SMOG index estimates the years of education required to understand text. However, because of analysing Greek text, i used an adaptation developed by Dritsa (2018). As she notes, applying the SMOG index to Greek lacks direct statistical validity, meaning that scores cannot be interpreted literally as years of schooling. Instead results must be red comparatively, so a speech A with higher score than speech B, means A is more complex to read, even if absolute values do not map onto education levels (Dritsa, 2018).

Following this approach, each speech was divided into three equal parts, from which a 30-sentence sample was extracted (10 from each section, centered). The number of polysyllabic words then computed using the syllabify() function from Greek-accentuation package, and the SMOG index was calculated using this formula: $\text{SMOG} = 1.0430 \times \sqrt{\frac{\text{polysyllables} \times 30}{\text{sentences}}} + 3.1291$. The resulting score provides a speech level estimate of linguistic complexity.

While SMOG index offers a simple and transparent measure, it is limited because it considers only sentence length and the number of polysyllabic words. As Hjorth (2024)

notes in his study, standard readability metrics do not account for whether words are common or rare, nor do they capture specialized rhetorical vocabulary. Ideally, an alternative measure would estimate complexity by incorporating word frequency distributions, for instance proportion of uncommon words, by contextualizing change in rhetorical style over a politician's career, capturing the transition from opposition to government and vice versa Hjorth (2024). By comparison, my approach is more constrained, focusing only on surface-level readability and thus missing deeper dimensions of rhetorical sophistication.

3.4 Semantic Drift - Greek BERT

To explore whether the rhetorical style of government speeches in the Hellenic Parliament has shifted over time, I employed a vector-based semantic analysis using Greek-BERT [3], a monolingual transformer model pretrained specifically for modern Greek (Koutsikakis et al., 2020). The model follows the bert-base-uncased architecture and was trained for one million steps on batches of 512 tokens. Its pretraining corpora included European Parliament and Hellenic Proceedings, making its design especially well-suited for capturing the formal and institutional style of political speeches.

Speeches shorter than 20 tokens were excluded to ensure meaningful embeddings and analysis was restricted to government speeches. Given BERT's 512-token limit, speeches were truncated where necessary. For each text, I extracted the [CLS] token embedding (a 768-dimensional vector) as a sentence-level representation. In 2020, where government/opposition labels were missing, all speeches were included and while this introduces some inconsistency, the effect is likely minimal as results still capture temporal shifts in rhetorical style.

In line with distributional semantics (Hovy, 2022)[sect.5.3], embeddings place texts in a high dimensional vector space where semantically similar speeches are closer to one another. To quantify semantic change over time, I computed cosine similarity (distance) between vectors across years, which captures the degree of alignment between rhetorical styles. A cosine similarity of 1 indicates identical embeddings, while values closer to 0 suggest semantic divergence. This method allowed me to approximate whether "meaning

[3] https://huggingface.co/nlpaueb/bert-base-greek-uncased-v1

space" of government speeches shifts over time.

For this analysis, I used off-the-shelf embeddings without fine-tuning. This choice is supported by Rodriguez and Spirling (2022), who show that pretrained models often perform as well as or better than domain-specific embeddings in political science applications, particularly when the corpus is smaller than the pretraining data. Similarly, Terechshenko et al. (2020) argue that pretrained transformers substantially improve text representation by encoding not only words in isolation byt also their order and syntactic relations, which is crucial for studying rhetorical style.

4 Analysis

4.1 Emotional Rhetoric in Government & Opposition

In order to answer RQ1, I quantified emotional rhetoric in parliamentary speeches using a lexicon (dictionary)-based method. The analysis presents government vs, opposition contrast and temporal variation. Figure 4.1.1 compares the mean intensity of 6 basic emotions in government and opposition speeches. Specifically, both groups draw heavily on anger, disgust and surprise, while emotions such as happiness and sadness are less prominent. Government speeches exhibit higher mean scores across all six emotions, indicating a more varied emotional repertoire. By contrast, opposition speeches rely more on disgust and show comparable levels of anger, reflecting a more consistently confrontational tone. This pattern agrees also with Widmann (2021)'s argument that anger and disgust serve as effective tools for blaming and discrediting opponents, while governments must mobilize a wider range of emotions (like fear and hope), to legitimize their actions and decisions.

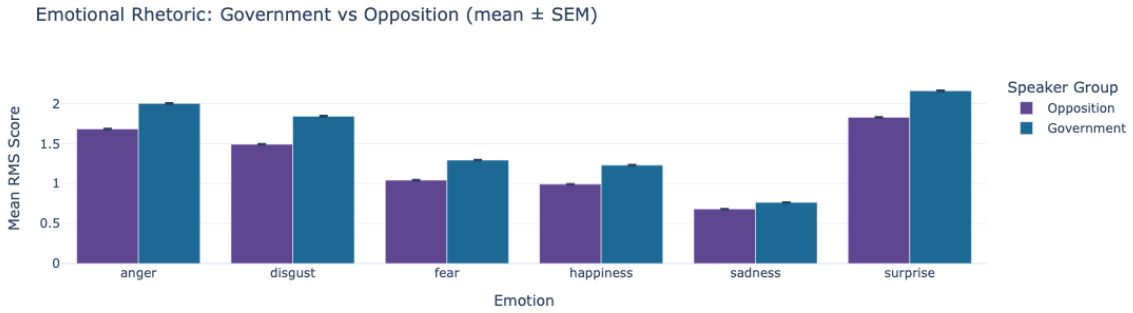


Figure 4.1.1: Mean RMS Score comparison of emotional rhetoric between government and opposition.

Additionally, Figure 4.1.2. presents how emotional rhetoric fluctuated over the decade in response to Greece's crises. Government speeches show peaks of emotional intensity during the sovereign debt crisis and near-bankruptcy (2010), the 2015 "Grexit" referendum and political turmoil, and again after 2018 with the emergence of the COVID-19 health crisis. A key political turning point was the 2015 election, when the radical left party *SYRIZA* defeated the traditional governing parties (*PASOK* and *Nea Dimokratia*). pushing them into opposition until *Nea Dimokratia* returned to power in 2019. This helps explain the mean score patterns: since *Nea Dimokratia* appears to have drawn more heavily on emotional rhetoric despite trying to cultivate a technocratic profile. Finally, the renewed increase after 2018 is likely partly attributable to the onset of the pandemic, which increased political and public stress.

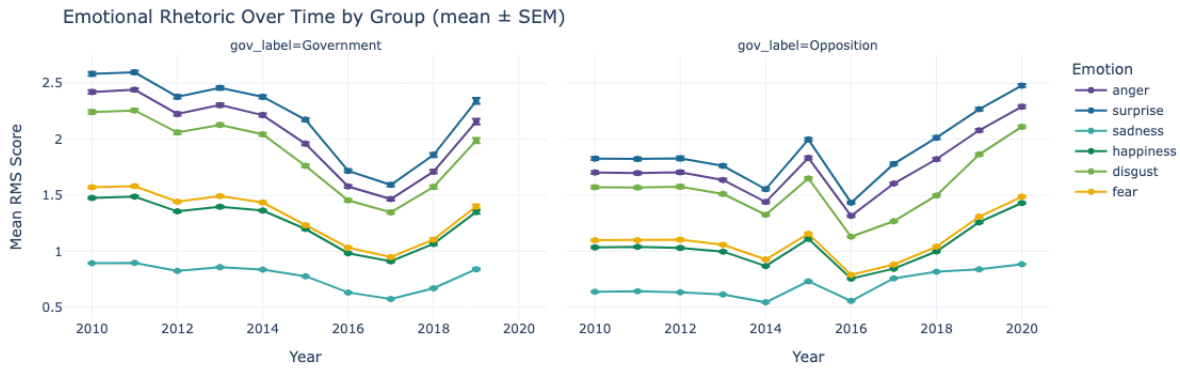


Figure 4.1.1: Emotional rhetoric between government and opposition over Time from 2010-2020 (Mean RMS Score).

4.2 Language Complexity of Political Elites

In order to answer RQ2, I calculated SMOG scores for opposition and government speeches from 2010-2020. As, shown on Figure 4.2.1 both groups cluster around values of 18-21, with opposition speeches tightly more centered and government ones exhibiting a slightly wider spread. This variation is likely influenced by the succession of short-lived governments and multiple leadership changes during the period. On average government speeches also tend to be more complex than those of opposition, especially in the first 5 years. After the turning point of 2015 and the election of radical left party *SYRIZA*, however, complexity drops sharply. This aligns with accusations that *SYRIZA* employed more populist, simplified language and arguments. Complexity rises again after 2019, when *Nea Dimokratia* -associated with a more technocratic profile- returned to government (Figure 4.2.2.). However, because government speeches are fewer overall, annual averages should be interpreted with caution, especially in later years when government speech counts drop (Figure 4.2.3).

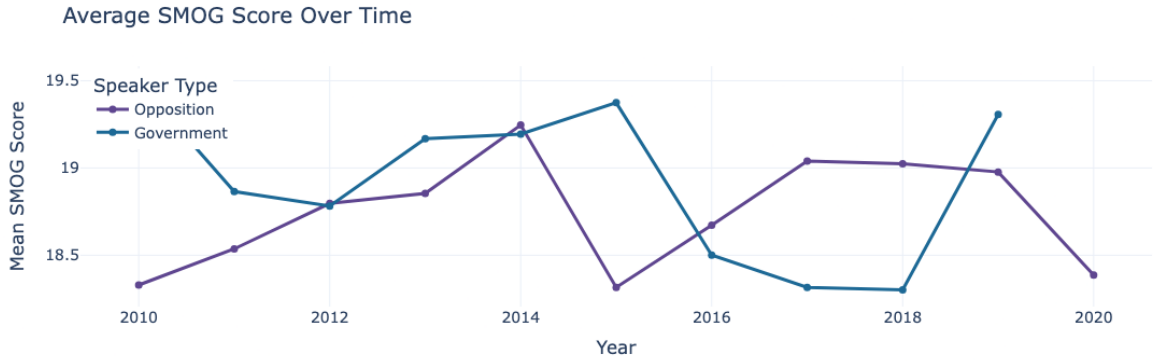


Figure 4.2.2: Average SMOG Score over time (2010-2020) comparison between government and opposition.

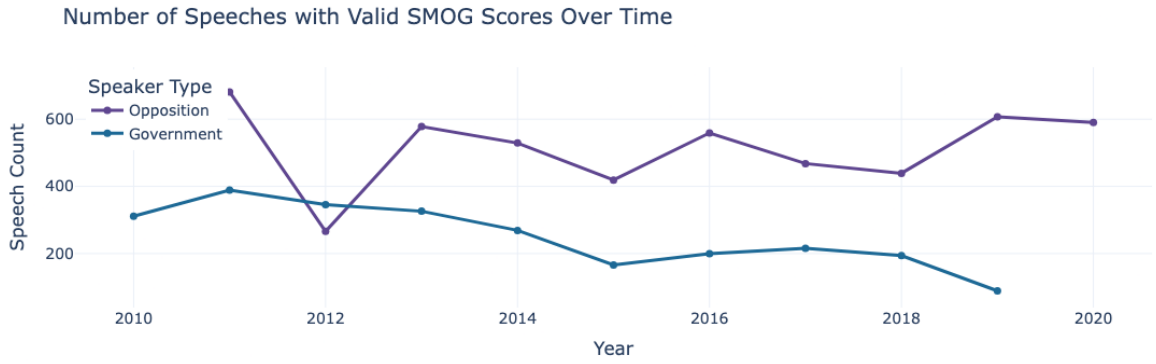


Figure 4.2.3: Number of Speeches with valid SMOG Score over time (2010-2020) comparison between government and opposition.

Overall, these results suggest that government officials do not consistently simply their rhetoric compared to opposition MPs. In this dataset while statistically significant differences were detected (T-statistic: 3.572, p-value: 0.0004 ; U-statistic: 7600343.500, p-value: 0.0053) the effect sizes was minimal (Cohen's d: 0.085). This indicates that readability as captured by the SMOG index does not meaningfully distinguish government from opposition rhetoric. As Hjorth (2024) notes, standard readability measures capture only surface-level variation and they overlook aspects like rhetorical strategy, specialized vocabulary, political role demands. My results therefore reinforce the limitations of applying the SMOG index for Greek parliamentary speech.

4.3 Temporal Shifts in Rhetorical Style

In order to answer RQ3, I calculated the temporal cosine distance between government speeches, where values close to 0 indicate stability while larger values signal rhetorical change. Although absolute cosine distances were minimal (0 to 0.006), this may reflect the fact that many Greek politicians share rhetorical styles due to comparable backgrounds and training. For most of the decade shifts are modest, with the exception of a pronounced peak in 2016 (Figure 4.3.1.), when rhetorical style diverged most sharply compared to previous years. In terms of cumulative drift (Figure 4.3.2.) relative to 2010 baseline, government rhetoric shows a gradual increase during early crisis years, reaching its peak around 2015, followed by partial reversion after 2016.

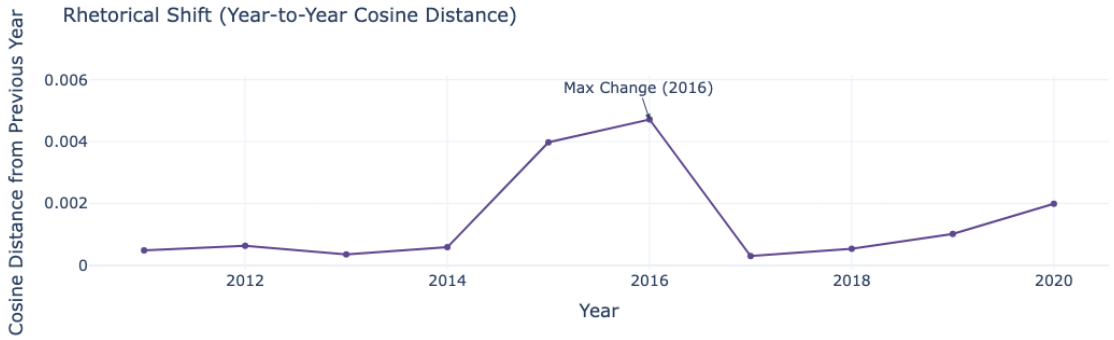


Figure 4.3.1.: Rhetorical Shift (Year-to-Year) - Cosine Distance from previous year.

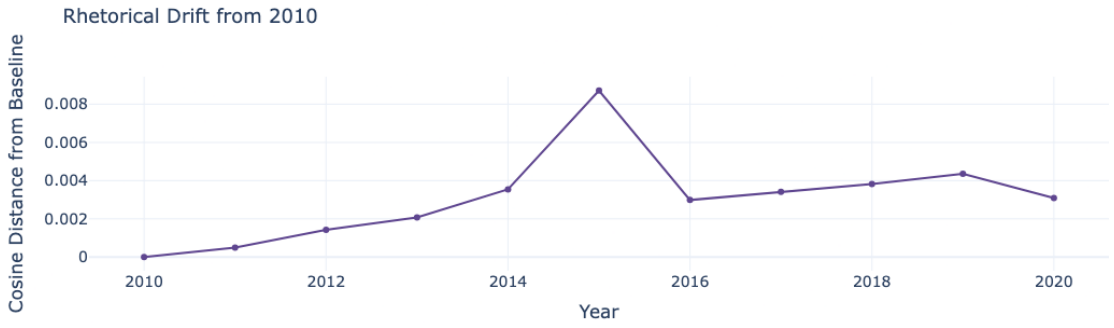


Figure 4.3.2.: Rhetorical Drift from 2010 - Cosine Distnace from baseline.

These results align with the findings of RQ1 and RQ2, since the sharp rhetorical break coincides with the turbulent period of 2015, marked by the rise of radical left *SYRIZA* government and the bailout referendum. This suggests that not only government turnover but also crisis conditions trigger noticeable changes in rhetorical framing. Importantly, after this dramatic change, government rhetoric did not return to is original baseline. One explanation is that *Nea Dimokratia* in preparation for its electoral victory, cultivated a more technocratic profile and rebranded itself under new leadership. These findings support with Rodriguez and Spirling (2022) observation that pretrained embeddings can capture even minimal but meaningful semantic shifts in political texts, and align with theories of rhetorical adaptation under crisis conditions (Widmann, 2021).

5 Conclusion

In this assignment, I analyzed the rhetorical style of government speeches in the Greek Parliament from 2010 to 2020, addressing three questions on emotional rhetoric, linguistic complexity, and rhetorical change over time. A range of methods covered in the ASDS II course were applied: lexicon (dictionary) based sentiment detection, readability index, and word embeddings, aiming to reflect the course's emphasis on combining computational techniques with social sciences reasoning.

The results highlight both the strengths and limitations of each method. Lexicon-based analysis (RQ1) captured contrasts in emotional rhetoric between government and opposition, yet the approach is constrained by dictionary coverage and challenges of Greek morphology. The SMOG index adapted for Greek (RQ2) provided a quantitative measure of rhetorical complexity, but the effect sizes were minimal, showing the narrow scope of readability metrics and the need for richer approaches to complexity. Word embeddings (RQ3) demonstrated the power of distributional semantics to detect subtle rhetorical drift over time, even without fine-tuning. However, Greek-BERT was pretrained on corpora overlapping with my dataset, reducing its ability to capture novelty. A further lesson concerns the role of preprocessing and dataset quality, since specific decisions and trade-offs in granularity strongly shape the outcomes and validity of the analysis.

To sump up, it is clear how methodological choices shape what we learn form text data. By applying multiple approaches, I was able to understand both the promises and limits of computational text analysis, while also reflecting on the role of researcher judgment in ensuring robust and meaningful insights.

References

Benoit, K., Munger, K., and Spirling, A. (2019). Measuring and explaining political sophistication through textual complexity. *American Journal of Political Science*, 63(2):491–508.

Denny, M. J. and Spirling, A. (2018). Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. *Political analysis*, 26(2):168–189.

Dritsa, K. (2018). Speech quality and sentiment analysis on the hellenic parliament proceedings. In *Master's thesis, Athens University of economics and business, Athens*.

Hjorth, F. (2024). Losing touch: The rhetorical cost of governing. *Comparative Political Studies*, page 00104140251349646.

Hovy, D. (2021). *Text Analysis in Python for Social Scientists: Discovery and Exploration*. Elements in Quantitative and Computational Methods for the Social Sciences. Cambridge University Press.

Hovy, D. (2022). *Text Analysis in Python for Social Scientists: Prediction and Classification*. Elements in Quantitative and Computational Methods for the Social Sciences. Cambridge University Press, Cambridge.

Koutsikakis, J., Chalkidis, I., Malakasiotis, P., and Androutsopoulos, I. (2020). Greek-bert: The greeks visiting sesame street. In *11th Hellenic conference on artificial intelligence*, pages 110–117.

Loukas, L., Smyrnioudis, N., Dikonomaki, C., Barbakos, S., Toumazatos, A., Koutsikakis, J., Kyriakakis, M., Georgiou, M., Vassos, S., Pavlopoulos, J., et al. (2024). Gr-nlp-toolkit: An open-source nlp toolkit for modern greek. *arXiv preprint arXiv:2412.08520*.

Pancer, S. M., Hunsberger, B., Pratt, M. W., Boisvert, S., and Roth, D. (1992). Political roles and the complexity of political rhetoric. *Political Psychology*, pages 31–43.

Proksch, S.-O., Lowe, W., Wäckerle, J., and Soroka, S. (2019). Multilingual sentiment analysis: A new approach to measuring conflict in legislative speeches. *Legislative Studies Quarterly*, 44(1):97–131.

Rodriguez, P. L. and Spirling, A. (2022). Word embeddings: What works, what doesn't, and how to tell the difference for applied research. *The Journal of Politics*, 84(1):101–115.

Terechshenko, Z., Linder, F., Padmakumar, V., Liu, M., Nagler, J., Tucker, J. A., and Bonneau, R. (2020). A comparison of methods in political science text classification: Transfer learning language models for politics. *Available at SSRN 3724644*.

Widmann, T. (2021). How emotional are populists really? factors explaining emotional appeals in the communication of political parties. *Political psychology*, 42(1):163–181.

A AI-Statement

ChatGPT by OpenAI (https://chatgpt.com) was employed to support proofreading process by offering suggestions, correcting grammatical and syntactic errors, and improving overall structure and formatting. It also provided guidance on adapting existing research code into python versions, building custom functions, and setting up the environment. In addition, it assisted in resolving code errors, library conflicts, and versions discrepancies, ensuring a stable and functional workflow.

B Python Code

Data Handling & Preprocessing

```
#data-cleaning-processing-core-libs
import pandas as pd
import numpy as np
from tqdm.notebook import tqdm
import time
tqdm.pandas()
import ipywidgets as widgets
widgets.IntSlider()
#text-processing
import re
```

```
import string
import unicodedata
#greek-nlp-toolkit-Loukas(2024)
from gr_nlp_toolkit import Pipeline
#greek-stopwords
import stopwordsiso as stopwords_iso
greek_stopwords = list(stopwords_iso.stopwords("el"))
```

```
df = pd.read_csv('par10_20c.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 536446 entries, 0 to 536445
Data columns (total 11 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   member_name            522524 non-null  object
 1   sitting_date           536446 non-null  object
 2   parliamentary_period   536446 non-null  object
 3   parliamentary_session  536446 non-null  object
 4   parliamentary_sitting  536446 non-null  object
 5   political_party        536327 non-null  object
 6   government             536446 non-null  object
 7   member_region          513121 non-null  object
 8   roles                  522524 non-null  object
 9   member_gender          522524 non-null  object
 10  speech                 536446 non-null  object
dtypes: object(11)
memory usage: 45.0+ MB
```

```
columns_to_drop = ["parliamentary_session", "parliamentary_sitting", "
    parliamentary_period","member_region"]
df = df.drop(columns=columns_to_drop)
```

```
missing_percent = df.isnull().mean().sort_values(ascending=False) * 100
print(missing_percent.round(2))
```

```
member_name          2.60
roles                2.60
member_gender        2.60
political_party      0.02
sitting_date         0.00
government           0.00
speech               0.00
dtype: float64
```

```
df = df.dropna(subset=["member_name", "roles", "member_gender", "
    political_party"])
df['sitting_date'] = pd.to_datetime(df['sitting_date'], errors='coerce'
    )
df['year'] = df['sitting_date'].dt.year
```

```
# Function to check if a speaker was in government
def is_government(row):
    gov_name = row['government'].lower()
    party = row['political_party'].lower()
    year = row['year']

    for coalition in gov_coalitions:
        if (coalition['pm'] in gov_name) and (year in coalition['years'
            ]) and (party in coalition['parties']):
             return 1
    return 0

    df['is_government'] = df.apply(is_government, axis=1)
```

```
def extract_speaker_role(role_text):
    role_text = str(role_text).lower()
    for title in gov_titles:
        if title in role_text:
            return title
    return None

    df['speaker_gov_role'] = df['roles'].apply(extract_speaker_role)
```

```
def extract_leadership_role(role_text):
```

```
    role_text = str(role_text).lower()
    for title in leadership_titles:
        if title in role_text:
            return title
    return None

    df['leadership_role'] = df['roles'].apply(extract_leadership_role)
```

```
df.to_csv("processed01_par10-20.csv", index=False)
```

```
p_df = pd.read_csv('processed01_par10-20.csv')
p_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 341805 entries, 0 to 341804
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         341805 non-null  int64
 1   member_name        341805 non-null  object
 2   sitting_date       341805 non-null  object
 3   political_party    341805 non-null  object
 4   government         341805 non-null  object
 5   roles              341805 non-null  object
 6   member_gender      341805 non-null  object
 7   speech             341805 non-null  object
 8   year               341805 non-null  int64
 9   is_government      341805 non-null  int64
 10  speaker_gov_role   60473 non-null   object
 11  leadership_role    10689 non-null   object
 12  speech_clean       341668 non-null  object
dtypes: int64(3), object(10)
memory usage: 33.9+ MB
```

```
#removing ntonation/diacritics (accent marks) used in written greek
def remove_greek_diacritics(text):
    text = str(text)
```

```
    text = unicodedata.normalize("NFD", text)
    text = ''.join([char for char in text if not unicodedata.combining(
        char)])
    return text
```

```
def basic_cleaning(text):
    # Remove Greek diacritics
    text = remove_greek_diacritics(text)

    # Lowercase
    text = text.lower()

    # Remove formal expressions and honorifics
    patterns_to_remove = [
     #this part of the code is regex containing greek characters - is
         missing here due to LaTex code conflicts with greek letters
    ]
    for pattern in patterns_to_remove:
        text = re.sub(pattern, '', text)

    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

```
#stopwords_removal-function
def stopwords_removal(text):
    #remove stopwords
    words = text.split()
    words = [word for word in words if word not in greek_stopwords]
    return ' '.join(words)
```

```
p_df['speech_clean'] = p_df['speech'].progress_apply(basic_cleaning)
```

Analysis RQ1 - 6 Sentiment Lexicon (Dritsa, 2018)

```
from collections import defaultdict
import jellyfish
import spacy

rq1_df = pd.read_csv('processed01_par10-20.csv',index_col=0)
```

```
nlp = spacy.load("el_core_news_sm")

def tokenize_and_lemmatize(text):
    doc = nlp(text)
    return [token.lemma_ for token in doc if not token.is_space]
```

```
# Load lexicon into dictionary
lexicon_df = pd.read_csv("out_lexicon_6sent.csv")

lexicon = {
    row['term'].strip(): [
        float(row['anger']),
        float(row['disgust']),
        float(row['fear']),
        float(row['happiness']),
        float(row['sadness']),
        float(row['surprise'])
    ]
    for _, row in lexicon_df.iterrows()
}
```

```
def sent6_vec(text, lexicon, tokenize_fn):
    word_vecs = []

    for word in tokenize_fn(text):
        if word in lexicon:
            word_vecs.append(lexicon[word])

    if not word_vecs:
        return [0] * 6

    word_vecs = np.array(word_vecs)
```

```
    rms = np.sqrt(np.mean(np.square(word_vecs), axis=0))
    return [round(v, 3) for v in rms]
```

```
rq1_df['sent6_vec'] = rq1_df['speech_clean'].progress_apply(
    lambda text: sent6_vec(text, lexicon, tokenize_and_lemmatize)
)
```

```
sentiment_cols = ['anger', 'disgust', 'fear', 'happiness', 'sadness', '
    surprise']
sentiment_redf = pd.DataFrame(rq1_df['sent6_vec'].tolist(), columns=
    sentiment_cols)

meta_cols = ['member_name', 'political_party', 'year', 'is_government',
     'roles']
meta_df = rq1_df[meta_cols].reset_index(drop=True)

rq1_results_df = pd.concat([meta_df, sentiment_redf], axis=1)
```

Analysis RQ2 - SMOG Index Greek Dritsa, (2018)

```
rq2_df = pd.read_csv('processed01_par10-20.csv',index_col=0)
```

```
rq2_df = pd.read_csv('processed01_par10-20.csv',index_col=0)
```

```
def prepare_text_for_smog(text):
    if not isinstance(text, str):
        return None

    #Remove Greek diacritics
    text = remove_greek_diacritics(text)

    #Normalize to lowercase
    text = text.lower()

    #Keep sentence punctuation: periods (.), question marks (?),
       exclamation (!)
    #Replace them all with periods for SMOG sentence splitting
    text = text.replace('!', '.').replace('?', '.')

```

```
    #Remove formal/honorific expressions
    patterns_to_remove = [
    #this part of the code is regex containing greek characters - is missing here due to LaTex code conflicts with greek letters
    ]
    for pattern in patterns_to_remove:
        text = re.sub(pattern, '', text)

    #Convert digits to text using invoice library
    def normalize_numbers(match):
        try:
            return invoice.num_to_text(int(match.group()))
        except:
            return match.group()

    text = re.sub(r'\d+', normalize_numbers, text)

    #Remove unwanted symbols (keep Greek letters, spaces, periods)
    text = re.sub(r'[#this part of the code is regex containing greek characters - is missing here due to LaTex code conflicts with greek letters]+', ' ', text)

    #Collapse multiple spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

```
rq2_df['speech_for_smog'] = rq2_df['speech'].progress_apply(prepare_text_for_smog)
```

```
rq2_df.to_csv('processed0rq2_par10-20.csv.csv', index=False)
```

```
!pip install greek-accentuation invoice nltk
```

```
import math
import spacy
from greek_accentuation.syllabify import syllabify
import invoice
```

```
nlp = spacy.load("el_core_news_sm")
```

```python
#Function to compute SMOG index - Python version based on Dritsa's code
def compute_smog(text):
    if not isinstance(text, str):
        return None

    #Greek sentence tokenizer
    sentences = [sent.text for sent in nlp(text).sents]

    #SMOG needs at least 60 sentences to sample 30 | Dritsa 2018 followed
    if len(sentences) < 60:
        return None

    #get 20 middle sentences from a chunk
    def middle_20(sent_list):
        center = len(sent_list) // 2
        return sent_list[center - 10 : center + 10]

    #sample 10 from each third of the speech
    part1 = middle_20(sentences[:len(sentences)//3])
    part2 = middle_20(sentences[len(sentences)//3:2*len(sentences)//3])
    part3 = middle_20(sentences[2*len(sentences)//3:])
    sample_sentences = part1 + part2 + part3

    polysyllables = 0

    for sentence in sample_sentences:
        words = sentence.split()  # token-level accuracy is not critical for SMOG
        for word in words:
            if word.isalpha():
                try:
                    if len(syllabify(word)) > 2:
                        polysyllables += 1
                except:
                    continue

    # SMOG formula (based on 30-sentence sample)
```

```
    smog = 1.0430 * math.sqrt((polysyllables * 30) / len(
        sample_sentences)) + 3.1291
    return round(smog, 2)
```

```
rq2_df['smog_score'] = rq2_df['speech_for_smog'].progress_apply(
    compute_smog)
```

```
columns_to_keep = [
    'member_name', 'year', 'political_party', 'is_government',
    'speaker_gov_role', 'leadership_role', 'roles', 'smog_score'
]
rq2_results_df = rq2_df[columns_to_keep].copy()
rq2_results_df.to_csv('RQ2_results_df.csv', index=False)
```

Analysis RQ3 - Semantic Drift - Greek-BERT

```
import pickle, os
from pathlib import Path
import hashlib
from sklearn.metrics.pairwise import cosine_similarity
```

```
rq3_df = pd.read_csv('processed01_par10-20.csv')
rq3_df = rq3_df[rq3_df['speech_clean'].notnull()]
rq3_df = rq3_df[rq3_df['speech_clean'].str.split().str.len() > 50]
gov_df = rq3_df[rq3_df['is_government'] == 1]

gov_df.to_csv('processed0rq3_par10-20.csv',index=False)
```

```
gov_df.to_csv('processed0rq3_par10-20.csv',index=False)
gov_df = pd.read_csv('processed0rq3_par10-20.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 44997 entries, 13 to 316444
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   member_name       44997 non-null  object
 1   sitting_date      44997 non-null  object
 2   political_party   44997 non-null  object
```

```
 3   government          44997 non-null  object
 4   roles               44997 non-null  object
 5   member_gender       44997 non-null  object
 6   speech              44997 non-null  object
 7   year                44997 non-null  int64
 8   is_government       44997 non-null  int64
 9   speaker_gov_role    21484 non-null  object
 10  leadership_role     1336 non-null   object
 11  speech_clean        44997 non-null  object
dtypes: int64(2), object(10)
memory usage: 4.5+ MB
```

```python
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer = AutoTokenizer.from_pretrained("nlpaueb/bert-base-greek-uncased-v1")
model = AutoModel.from_pretrained("nlpaueb/bert-base-greek-uncased-v1")
model.eval()
```

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(35000, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=
       True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0-11): 12 x BertLayer(
        (attention): BertAttention(
          (self): BertSdpaSelfAttention(
            (query): Linear(in_features=768, out_features=768,
               bias=True)
```

```
            (key): Linear(in_features=768, out_features=768, bias
               =True)
            (value): Linear(in_features=768, out_features=768,
               bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768,
               bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12,
               elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072,
             bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768,
             bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12,
             elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

```python
#Speech Embedding Function
def get_embedding(text, tokenizer, model, device='cpu'):
    # Tokenize the input text and return PyTorch tensors
    inputs = tokenizer(text, return_tensors="pt", truncation=True,
        max_length=512)
    # Move inputs to the device (e.g., CPU or CUDA/GPU)
    inputs = {key: val.to(device) for key, val in inputs.items()}
    # Disable gradient tracking (for efficiency)
    with torch.no_grad():
        outputs = model(**inputs)
    # Get the [CLS] token embedding (first token)
    cls_embedding = outputs.last_hidden_state[:, 0, :]  # shape: [1,
        768]
    # Convert from PyTorch tensor to NumPy array
    return cls_embedding.squeeze().cpu().numpy()
```

```python
year_embeddings = {}

for year, group in gov_df.groupby("year"):
    print(f"Processing year {year} with {len(group)} speeches...")

    embeddings = []

    for speech in tqdm(group["speech_clean"], desc=f"Year {year}"):
        try:
            vec = get_embedding(speech, tokenizer, model)
            embeddings.append(vec)
        except Exception as e:
            continue  # Skip any speech that causes an error

    if embeddings:
        year_embeddings[year] = np.mean(embeddings, axis=0)
```

```python
import pickle, os
from pathlib import Path
```

```python
Path("embeddings").mkdir(exist_ok=True)

with open("embeddings/year_embeddings_gov.pkl", "wb") as f:
```

```
    pickle.dump(year_embeddings, f)

years_sorted = sorted(year_embeddings)
matrix = np.vstack([year_embeddings[y] for y in years_sorted])
np.savez_compressed("embeddings/year_embeddings_gov.npz",
                    years=np.array(years_sorted), vectors=matrix)

print("Saved to: embeddings/year_embeddings_gov.pkl and .npz")
```

```
import hashlib
```

```
# --- tiny per-speech cache ---
CACHE_DIR = "embeddings/speech_cache"
os.makedirs(CACHE_DIR, exist_ok=True)

def embed_with_cache(text):
    key = hashlib.md5(text.encode("utf-8")).hexdigest()
    path = os.path.join(CACHE_DIR, f"{key}.npy")
    if os.path.exists(path):
        return np.load(path)
    vec = get_embedding(text, tokenizer, model)
    np.save(path, vec)
    return vec

# --- select 2020 speeches (ALL, not just gov) ---
g20 = rq3_df[(rq3_df["year"] == 2020) & rq3_df["speech_clean"].notnull
    ()].copy()

# use a gentler length filter to avoid dropping many 2020 speeches
g20 = g20[g20["speech_clean"].str.split().str.len() > 20]

print("2020 speeches to embed:", len(g20))

# --- compute 2020 average embedding ---
embeds_2020 = [embed_with_cache(s) for s in tqdm(g20["speech_clean"],
    desc="Year 2020 (ALL speeches)")]
if embeds_2020:
    year_embeddings[2020] = np.mean(embeds_2020, axis=0)
    print("2020 vector shape:", year_embeddings[2020].shape)
```

```
else:
    print("No embeddings collected for 2020.")
```

```
2020 speeches to embed: 6749
Error displaying widget: model not found
2020 vector shape: (768,)
```

```
with open("embeddings/year_embeddings_gov.pkl", "wb") as f:
    pickle.dump(year_embeddings, f)

years_sorted = sorted(year_embeddings)
matrix = np.vstack([year_embeddings[y] for y in years_sorted])
np.savez_compressed("embeddings/year_embeddings_gov.npz",
                    years=np.array(years_sorted), vectors=matrix)

print("Years embedded:", years_sorted)
print("Example vector shape:", year_embeddings[years_sorted[0]].shape)
```

```
Years embedded: [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
   2018, 2019, 2020]
Example vector shape: (768,)
```

```
npz_path = Path("embeddings/year_embeddings_gov.npz")
pkl_path = Path("embeddings/year_embeddings_gov.pkl")

if npz_path.exists():
    npz = np.load(npz_path, allow_pickle=True)
    years_sorted = npz["years"].astype(int)
    matrix = npz["vectors"]  # shape: (n_years, 768)
    # Also reconstruct dict (handy later)
    year_embeddings = {int(y): vec for y, vec in zip(years_sorted,
        matrix)}
else:
    # Fallback: load the pickle dict
    with open(pkl_path, "rb") as f:
        year_embeddings = pickle.load(f)
    years_sorted = np.array(sorted(year_embeddings))
    matrix = np.vstack([year_embeddings[int(y)] for y in years_sorted])

```

```
print("Years:", years_sorted.tolist())
print("Matrix shape:", matrix.shape)  # (n_years, 768)
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Cosine similarity between all years (n_years x n_years)
cos_sim = cosine_similarity(matrix)
# Convert to distances (0 = identical, larger = more different)
cos_dist = 1 - cos_sim

# Put in a labeled DataFrame (handy to inspect or export)
dist_df = pd.DataFrame(cos_dist, index=years_sorted, columns=
    years_sorted)

# Year-over-year distance (how much each year moves from the previous
    one)
yoy = []
for i in range(1, len(years_sorted)):
    prev_i = i - 1
    yoy.append({
        "year": int(years_sorted[i]),
        "from_year": int(years_sorted[prev_i]),
        "yoy_distance": float(cos_dist[i, prev_i])
    })
yoy_df = pd.DataFrame(yoy)

# Distance from the first year (cumulative drift)
baseline_year = int(years_sorted[0])
drift = []
for i in range(len(years_sorted)):
    drift.append({
        "year": int(years_sorted[i]),
        "from_year": baseline_year,
        "drift_distance": float(cos_dist[i, 0])
    })
drift_df = pd.DataFrame(drift)

# Save for your appendix
dist_df.to_csv("embeddings/year_pairwise_cosine_distance.csv", index=
```

```
    True)
yoy_df.to_csv("embeddings/year_yoy_distance.csv", index=False)
drift_df.to_csv("embeddings/year_drift_from_baseline.csv", index=False)

print("Saved: pairwise, YoY, and drift CSVs in /embeddings/")
yoy_df.head(), drift_df.head()
```