Marc E. Canby
April 24, 2018
Word Count: 1598

Uncovering Hidden Meaning in the *De Re Publica*
A Computational Analysis of the Text

For centuries, classical scholars have analyzed Latin texts simply by reading them, which has allowed scholars to decipher meaning, find patterns, and discover historical information. However, certain features of the text can be difficult – and sometimes impossible – to ascertain by studying them in this way: for example, a problem as basic as determining the average number of words in a sentence can be difficult for humans reading the text to solve. With the advent of high-speed computers in the last half-century, such problems are now tractable, opening the door to information about the text that was not previously available.

In this paper, I will perform a computational analysis of Cicero's *De Re Publica*. I will emphasize an analysis that is not attainable simply by reading the text. I will first explain how I obtained the text on the computer and perform a preliminary analysis. Then, I will investigate *distributional word semantics*, which studies semantic relationships between words. Finally, I will perform *sentence clustering*, which groups sentences based on their meaning. This paper is designed to be accessible to students of Latin without little to no math and computer science background.

Section 1. Obtaining and Cleaning the Data

I choose to use the Python coding language[1], a popular choice for researchers in text analysis. I accessed a digital copy of the *De Re Publica* from the *Latin Library*, which is made up of sentences of the following form[2]:

```
['nempe', 'ab', 'iis', 'qui', 'haec', 'disciplinis', 'informata', 'alia',
'moribus', 'confirmarunt', ',', 'sanxerunt', 'autem', 'alia', 'legibus', '.']
```
[3, I.2]

I first cleaned several "messy" elements of the raw text. For example, consider the beginning of the first sentence:

```
['&', 'lt', ';', 'im&gt', ';', 'petu', 'liberavissent', ',', 'nec', ...]
```
[3, I.1]

Clearly, left and right angle brackets (< and >) are encoded as `&lt;` and `&gt;` respectively; I amended these to their proper symbol before proceeding. English titles such as

```
['Cicero', 'The', 'Latin', 'Library', 'Classics', 'Page']
```

also occurred in the text, which I removed before examining the data. Neglecting to clean the text in these ways could result in misleading conclusions.

Section 2. Exploratory Text Analysis

I begin with an exploratory analysis. I first compare the size of the text to the average size among Cicero's philosophical treatises[3]:

	De Re Publica	Average across Cicero's Philosophy
Number of characters	$109,777$	$136,893$
Number of words	$20,067$	$24,924$
Number of sentences	820	$1,059$

Figure 1: Size of Text

We see that the *De Re Publica* is shorter than Cicero's average philosophy work. I also analyze the part-of-speech distribution[4]:

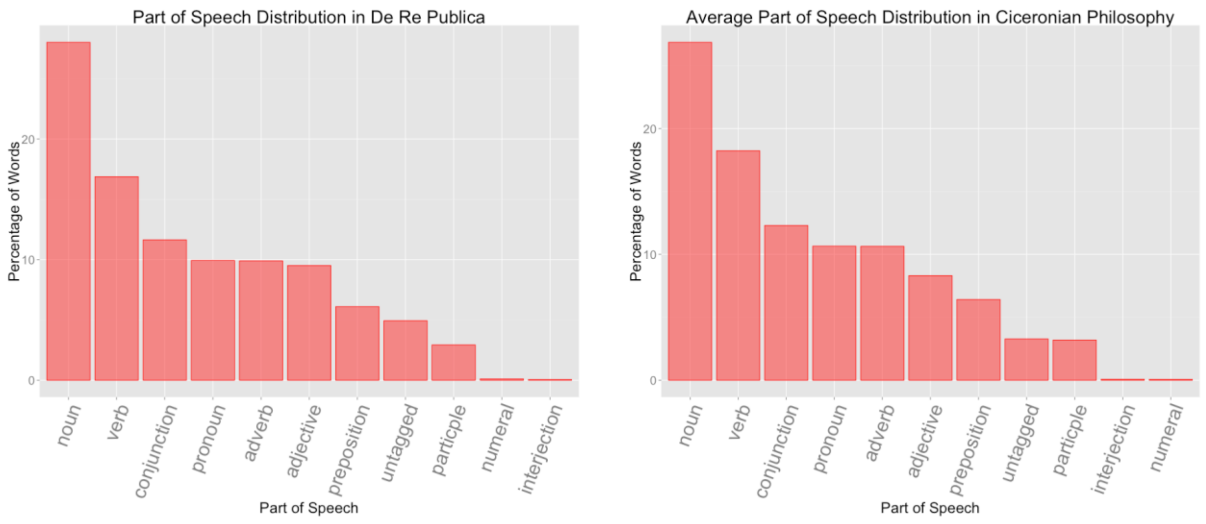

Figure 2: Part-of-speech Distribution

As expected, nouns and verbs are most common, followed by conjunctions. There does not appear to be a significant difference between the part-of-speech distributions of the *De Re Publica* and Cicero's other philosophical works, which is a naive indicator that this text does not deviate greatly from Cicero's "average" writing style.

Section 3. Distributional Word Semantics: Introduction

After exploring the data, I investigate distributional word semantics, which examines semantic relationships between linguistic items (such as words and sentences) in a text corpus. The goal is to encode each item as a vector (series of numbers) and analyze the relationship among the vectors. Consider Figure 3 below:

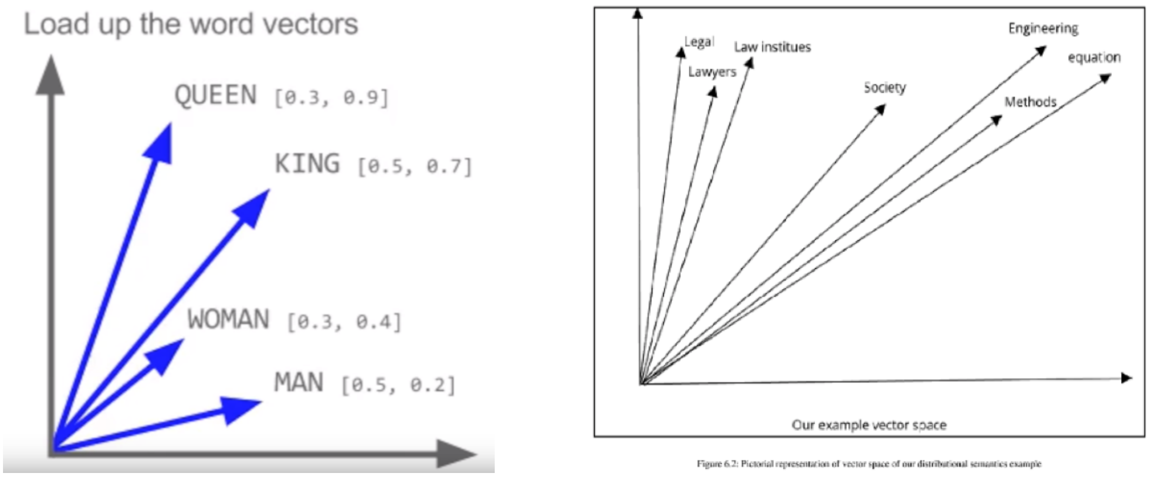

Figure 3: Distributional Word Semantics [1, 10]

The left figure shows the relationship between the words "king" and "queen": `queen = king - man + woman`. The right figure shows what might happen if we perform this analysis on a group of legal and scientific articles: legal terms such as "lawyers" and "law institutes" occur in more similar contexts than scientific terms like "engineering" and "equation".

Section 4. Distributional Word Semantics: Word2Vec Algorithm

To analyze distributional word semantics, I use the Word2Vec algorithm, which was invented by Google in 2013. Figure 4 shows an overview of the algorithm [10, 9]:

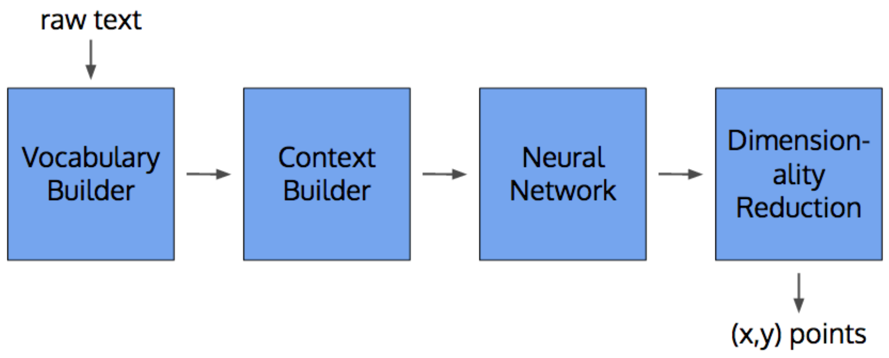

Figure 4: Overview of Word2Vec Algorithm

To illustrate how the algorithm works, I will show its effect on a single sentence:

```
 ['nempe', 'ab', 'iis', 'qui', 'haec', 'disciplinis', 'informata', 'alia',
'moribus', 'confirmarunt', ',', 'sanxerunt', 'autem', 'alia', 'legibus', '.']
```

Since declension and conjugation generally do not alter the meaning of words, we use the *base* *form*, or dictionary entry, of each word. We also remove *stop words*, which are words like *et* and *sum* that add little additional meaning to sentences. Retaining these words tends to mislead the algorithm. We would transform the example sentence as follows[5]:

```
['nempe', 'ab', 'iis', 'qui', 'haec', 'disciplinis', 'informata', 'alia',
'moribus', 'confirmarunt', ',', 'sanxerunt', 'autem', 'alia', 'legibus', '.']
                                   ⇓
['nempe', 'ab', 'is', 'qui', 'hic', 'disciplina', 'informo', 'alius2', 'mos',
          'confirmo', 'sancio', 'autem', 'alius2', 'lex']
                                   ⇓
['nempe', 'disciplina', 'informo', 'alius2', 'mos', 'confirmo', 'sancio',
                            'alius2', 'lex']
```

Next, we build *context pairs* for each word in the text: this represents every word by the words around it. Figure 5 shows the context pairs for our sample sentence (note that `'----'` denotes the beginning and end of a sentence):

`('----','disciplina')` → `'nempe'`	`('mos','sancio')` → `'confirmo'`
`('nempe','informo')` → `'disciplina'`	`('confirmo','alius2')` → `'sancio'`
`('disciplina','alius2')` → `'informo'`	`('sancio','lex')` → `'alius2'`
`('informo','mos')` → `'alius2'`	`('alius2','----')` → `'lex'`
`('alius2','confirmo')` → `'mos'`	

Figure 5: Context Builder Example

The entry `('nempe','informo')` → `'disciplina'`, for example, indicates that the word `'disciplina'` can be found between words `'nempe'` and `'informo'`.

The final step of the algorithm generates a vector for each word. The idea is that words with similar vectors appear in similar contexts. The details of this step are highly complex; Figure 6 provides an overview:

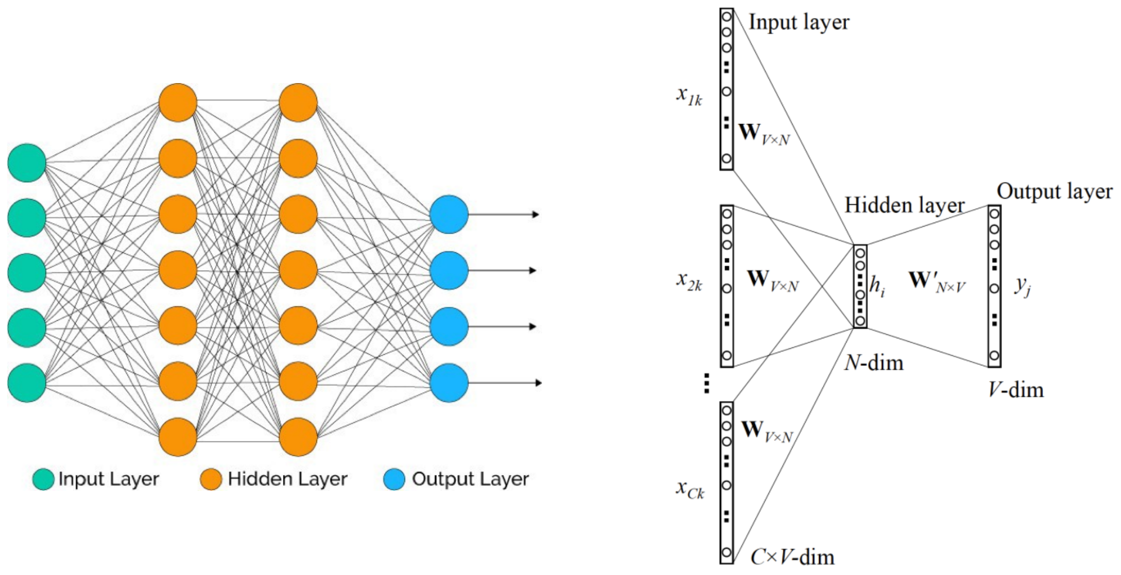

Figure 6: Neural Network in Word2Vec [5, 6]

The left figure illustrates a generic *neural network*; each circle is a neuron and each line connects neurons. This attempts to model the human brain: humans perceive multidimensional input (represented by the green circles) and make an educated decision about their perception (represented by one of the blue circles). The right image shows the neural network setup for Word2Vec; each rectangle on the input layer represents a context word, and each circle on the output layer represents a unique word in the text[6]. By observing context pairs, the network attempts to maximize the probability of predicting the correct word for a given context. Once optimized, the matrix $\boldsymbol{W}_{V\times N}$ contains the vectors for each word.

Section 5. Distributional Word Semantics: Results

We now present the results of running this algorithm on the *De Re Publica*. After reducing the size of the vectors to two dimensions[7], we obtain the plot shown in Figure 7:

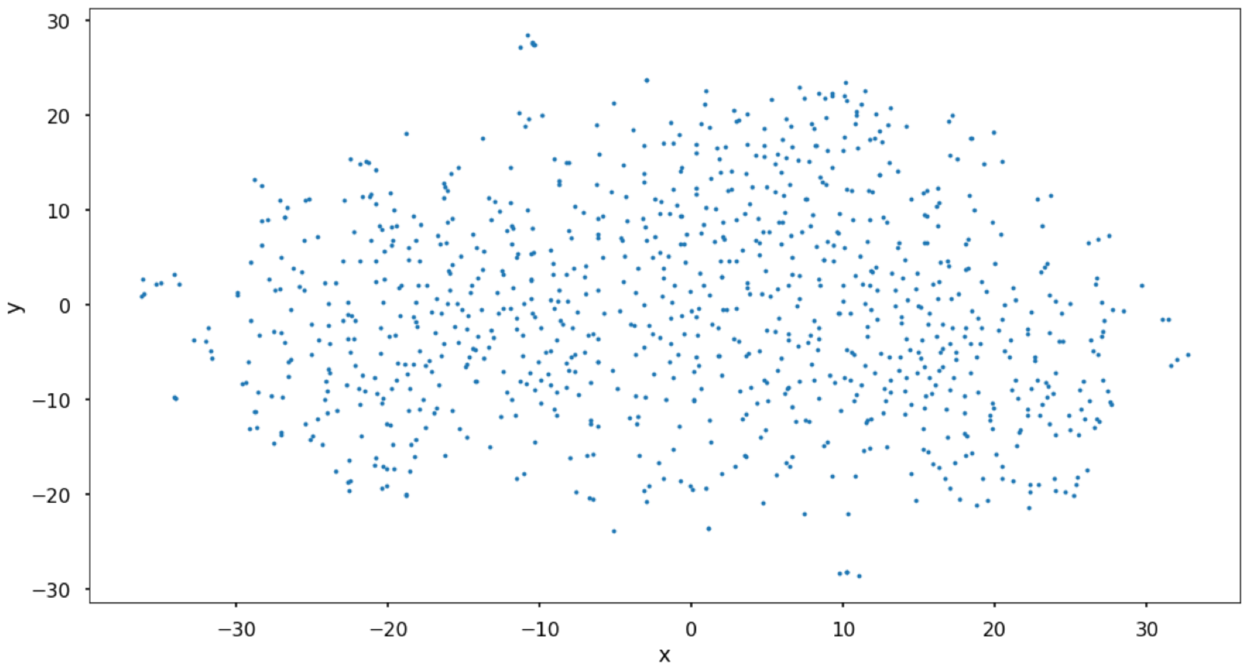

Figure 7: Word2Vec Results on the *De Re Publica*

Each point signifies a word, and nearby points represent words used in similar contexts. It is hard to ascertain much information from this figure, so we examine zoomed-in sections:

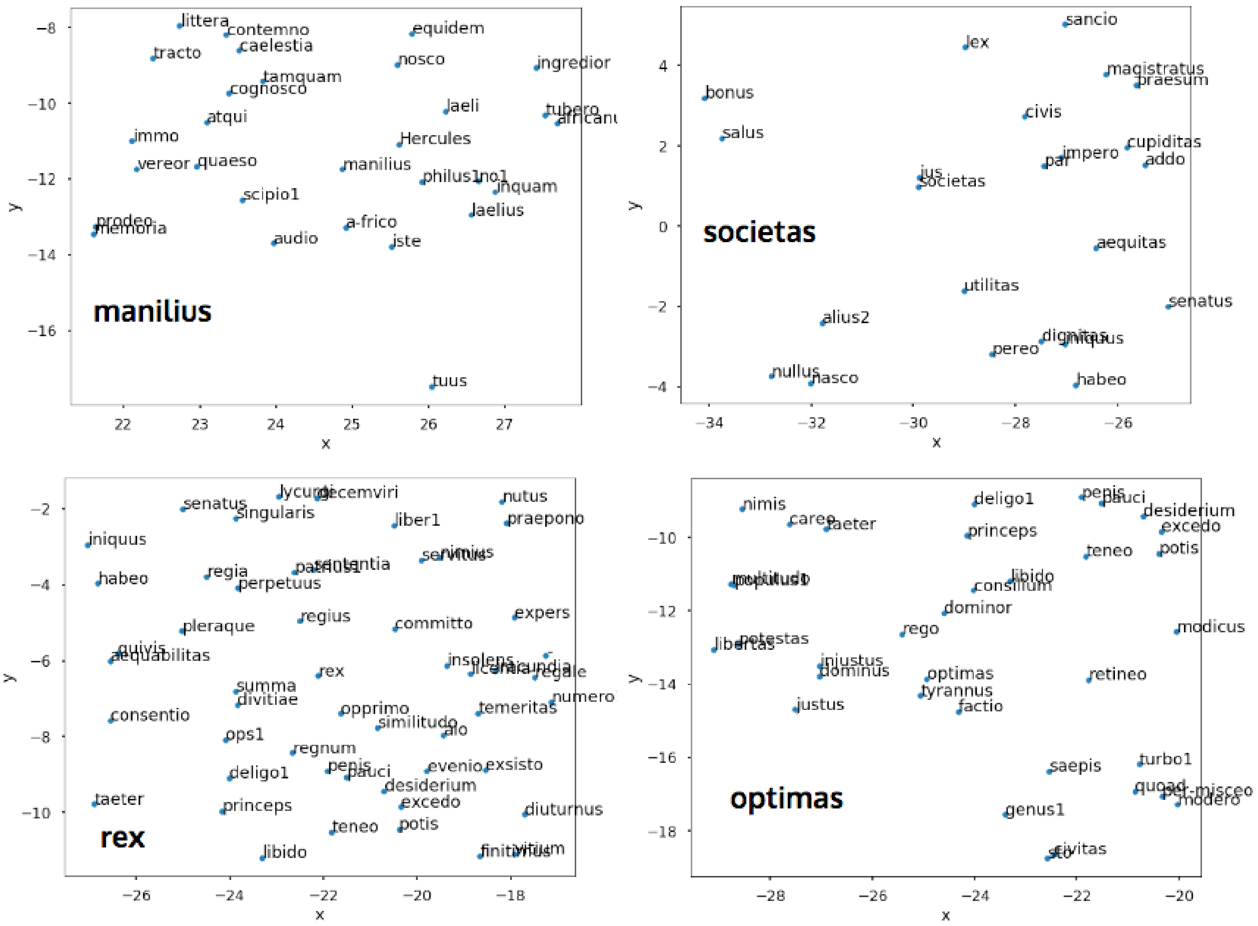

Figure 8: Zoomed Word2Vec Results on the *De Re Publica*

We see that Cicero uses names like *Manilius*, *Scipio* and *Philus* in similar contexts, which makes sense since these are often the subjects of speaking verbs. Similarly, words like *rex*, *opprimo*, and *regnum* appear near each other, which all relate to monarchical rule. An examination of the plots of *societas* and *optimas* shows similar patterns. Thus, this technique allows us to see patterns in Cicero's writing that are not necessarily evident when we read the text.

Section 6. Sentence Clustering: K-Means Algorithm

Next, I cluster sentences into K groups with the classic K-means algorithm. Figure 9 provides an overview [7, 10]:

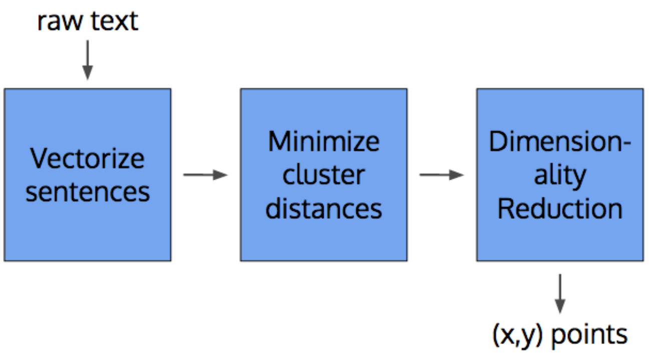

Figure 9: Overview of K-Means Algorithm

We first represent sentences as vectors; the distance between vectors will indicate sentence similarity. I use the *Term-Frequency-Inverse-Document-Frequency* representation, which encodes each sentence as a vector with length equal to the number of unique words in the text. The ith element of a sentence's vector denotes how many times the ith vocabulary term appears in that sentence. Since using pure counts would inevitably make longer sentences have larger entries, we normalize each sentence by its length, creating *term frequencies*. The result is demonstrated on two sample sentences below (as before, I use base forms and remove stop words):

```
['nempe', 'disciplina', 'informo', 'alius2', 'mos', 'confirmo', 'sancio',
'alius2', 'lex']
['oratio', 'ex-quaero', 'antepono', 'benus', 'constituo', 'civitas', 'publicus',
'jus', 'mos']
```
[3, I.2]

$$\Downarrow$$

```
[1,1,1,2,1,1,1,1,0,0,0,0,0,0,0,0]
[0,0,0,0,1,0,0,0,1,1,1,1,1,1,1,1]
```

$$\Downarrow$$

```
[1/9,1/9,1/9,2/9,1/9,1/9,1/9,1/9,0,0,0,0,0,0,0,0]
[0,0,0,0,1/9,0,0,0,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9]
```

The idea behind *inverse document frequencies* penalizes the values of words that appear in many sentences; for example, a word that appears in every sentence will be unlikely to help distinguish sentences, and its value would be set to 0 in every sentence. The mathematics behind this is omitted[8].

Next, we use the K-means algorithm to assign a group to each sentence. The algorithm is described below:

1. Assign K centroids (vectors) randomly, corresponding to the center of each group.
2. For each sentence, calculate the distance to each centroid, and assign the sentence to the group with the nearest centroid.
3. Calculate new centroids based on the sentence assignments, and repeat until convergence.

Section 7. Sentence Clustering: Results

After running this algorithm on the *De Re Publica* and reducing the dimensionality of the vectors, we get the following results:

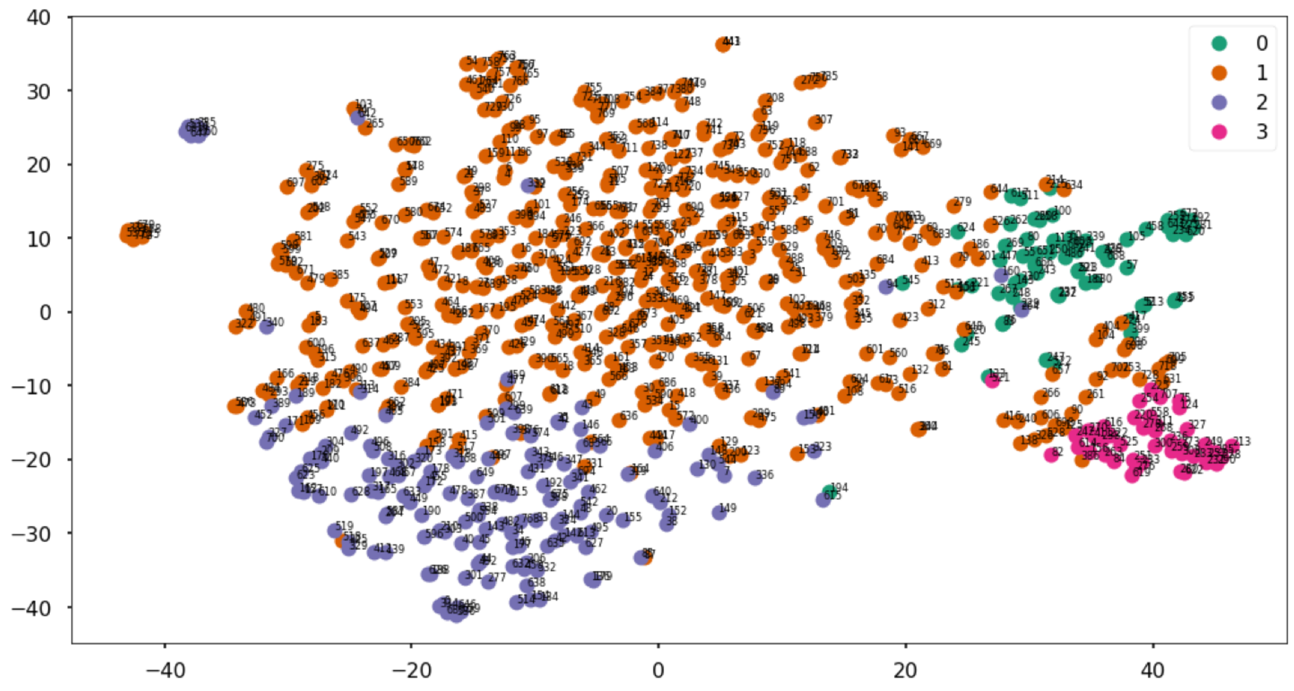

Figure 10: Sentence Clustering Results on the *De Re Publica*

Each point in Figure 10 represents a sentence, and its color indicates to which of four groups it belongs. We see a clear separation between groups, which means the sentences truly belong to distinct categories.

Figure 11 shows the words that are most indicative of group membership, and Figure 12 shows three sample sentences from each group[9]:

Cluster 0	Cluster 1	Cluster 2	Cluster 3
scipio1	homo	res	laelius
reor	magnus	publicus	inquam
inquam	eo1	populus1	verus
probo	verus	redeo	minimus
quaero	civitas	bonus	philus1
magnus	fero	genus1	nosco
censeo1	bonus	magnus	no1
sentio	ito	sto	saepis
rex	facio	eo1	probe
verus	populus1	cicero	sentio

Figure 11: Most Indicative Words by Cluster

Cluster	Sentences
0	• (scipio) ‘et talis est quaeque res publica, qualis eius aut natura aut voluntas qui illam regit [3, I.4]. • (scipio) ‘quid? [3, I.61] • (scipio) ‘ille vero, et magna quidem cum [3, I.23]
1	• tu enim quam celebritatem sermonis hominum aut quam expetendam consequi gloriam potes? [3, VI.20] • itaque ut tum carere rege, sic pulso tarquinio nomen regis audire non poterat. [3, II.52] • quare et tibi, publi, et piis omnibus retinendus animus est in custodia corporis nec iniussu eius, a quo ille est vobis datus, ex hominum vita migrandum est, ne munus humanum assignatum a deo defugisse videamini. [3, VI.15]
2	• quod ita cum sit, ex tritus primis generibus longe praestat mea sententia regium, regio autem ipsi praestabit id quod erit aequatum et temperatum ex tribus primis rerum publicarum modis. [3, I.69] • sed quoniam de re publica loquimur, sunt que inlustriora quae publice fiunt, quoniamque eadem est ratio iuris in utroque, de populi sapientia dicendum puto, et ut iam omittam alios: noster hic populus, quem africanus hesterno sermone a stirpe repetivit, cuius imperio iam orbis terrae tenetur, iustitia an sapientia est e minimo omnium maximus factus? [3, III.24] • hoc errore vulgi cum rem publicam opes paucorum, non virtutes tenere coeperunt, nomen illi principes optimatium mordicus tenent, re autem carent eo nomine. [3, I.51]
3	• (laelius) ‘adducor,’ inquit, ’et prope modum adsentior.’ [3, I.61] • tum laelius: ‘ego vero istud ipsum genus orationis quod polliceris expecto.’ [3, I.38] • (laelius) ‘ista vero’ inquit ‘adulta vix’. [3, I.58]

Figure 12: Sentences by Cluster

Clusters 0 and 3 clearly contain small sentences spoken by Scipio and Laelius respectively. It is harder to tell the difference between Clusters 1 and 2. Therefore, I searched for patterns in clustering across the entire text[10]: practically all of the passages about astronomy belong to Cluster 1, and quotes about forms of government are equally divided between Clusters 1 and 2. I should note that we do not necessarily expect to see obvious differences in the clusters; the fact that similar topics are assigned to different groups suggests that Cicero – perhaps subconsciously – alters his vocabulary in sentences about the same subject. This is an interesting insight about the text that may be hard to see when reading it.

Section 8. Challenges and Conclusions

Despite finding interesting results, I faced many challenges. For one, there has been very little research on computational analysis of ancient texts; most research has focused on modern languages. While the algorithms are not language-specific, results can vary widely on languages with different structures. For example, Latin, unlike English, is highly inflected, which means that its meaning is largely dependent on word endings as opposed to syntax. Thus, algorithms that rely on syntax, like Word2Vec, tend to perform worse on Latin than on English; consequently, I adjusted the algorithm's parameters to accommodate freer syntax. The K-means clustering algorithm, on the other hand, relies only on vocabulary that appears in each sentence, rendering it immune to syntactic issues. Thus, it is important to understand the underlying structure of the language when computationally analyzing text.

In my research, I ran over a dozen algorithms, examining topics such as keyword extraction, summarization, and text generation; further analyses could explore sentiment analysis and comparisons to other Latin authors. I chose to present distributional word semantics and text clustering because I feel that these results are not obvious from reading the text. Performing a computational analysis can give us a fresh perspective on a work that is over 2000 years old, offering us a view into Cicero's mind that we have not seen before.

Section 9. Endnotes

[1]All of my code, as well as other information, can be found at [2].

[2]I used the Classical Languages Toolkit (CLTK), a Python library with support for loading classical texts, performing morphological analyses, and running various algorithms [4].

[3]I used the works that *The Latin Library* deems to be Cicero's philosophical works for this analysis. A listing can be found at [3].

[4]I used pre-built functionality from the CLTK to determine parts of speech; I did not code a part-of-speech tagger from scratch [4]. I also used the `ggplot` package in the R coding language to produce these plots [11].

[5]Algorithms to identify stop words and *lemmatize* words (find the base forms of inflected words) are provided by the CLTK [4].

[6]This is called the Continuous Bag of Words (CBOW) architecture; another common architecture for Word2Vec is the Skip-gram. A discussion of the difference between the architectures can be found at [10]. I obtained better results on the CBOW architecture.

[7]I used the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm for dimensionality reduction. I also attempted Principle Component Analysis (PCA) and Independent Component Analysis (ICA), but had best results with t-SNE [8].

[8]Readers interested in TFIDF vectors can find more information at [10].

[9]Three sentences were selected at random from each group.

[10]The text with each sentence highlighted by group membership can be found at [2].

Section 10. References

[1] Allen. Word2vec - deep learning. `http://www.lifestyletrading101.com/word2vec-deep-learning/`, 2018.

[2] Marc E. Canby. De re publica computational analysis. `https://github.com/marccanby/DeRePublicaAnalysis`, 2018.

[3] M. Tullius Cicero. *De Re Publica*. The Latin Library. `http://www.thelatinlibrary.com/cicero/repub.shtml`.

[4] Kyle P. Johnson et al. Cltk: The classical languages toolkit. `https://github.com/cltk/cltk`, 2014–2017. 10.5281/zenodo.593336.

[5] Conor McDonald. Machine learning fundamentals (ii): Neural networks. `https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef`, 2017.

[6] Alex Minnaar. Word2vec tutorial part ii: The continuous bag-of-words model. `http://mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_II_The_Continuous_Bag-of-Words_Model.pdf`, 2015.

[7] Brandon Rose. Document clustering with python. `http://brandonrose.org/clustering`, 2015.

[8] Scikit-learn. Api reference: Sklearn.decomposition. `http://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition`, 2007-2017.

[9] Miguel Stephane. Build and visualize word2vec model with gensim. `https://github.com/MiguelSteph/word2vec-with-gensim`, 2017.

[10] Jalaj Thanaki. *Python Natural Language Processing*. Packt Publishing, 2017.

[11] Hadley Wickham. Ggplot2. `http://ggplot2.org/`, 2013.