1 Computational Linguistic Analysis

1.1 Scaling Up Corpus Linguistics

Corpus linguistics has entered a golden age, driven by both the amount and the range of language that is now available for linguistic analysis. Corpus data is able to represent a population's usage at scale, bypassing the limitations which made introspection so important in the 1950s. But this wide availability of language data requires that linguists have the methods available to analyze it. And while there has been a surge of advances in natural language processing and computational linguistics, these advances have become increasingly disconnected from corpus linguistics and linguistic theory. This Element brings natural language processing and corpus linguistics together, showing how computational models can be used to answer both **categorization** and **comparison** problems. These computational models are presented using five case studies that will be introduced in the next section, ranging from syntactic analysis to register analysis to corpus-based sociolinguistics.

The goal here is to show how to use these computational models, what linguistic questions they can answer, and why it is important to scale up corpus linguistics in this way. A linguist can use this Element to learn how to use natural language processing to answer linguistic questions they are already familiar with. And a computer scientist can use this Element to learn about the linguistic assumptions and limitations behind computational methods, matters that are too often disregarded within natural language processing itself.

A categorization problem is about assigning a predefined label to some piece of language. At the word level, this could involve asking whether a particular open-class word is a noun or a verb. At the sentence level, this could be asking what kind of construction a particular sentence represents. At the document level, this could be asking whether a particular speaker represents New Zealand English or Australian English. All of these questions can be answered using a **text classifier**. This is a type of supervised machine learning in which we as linguists define the categories that we are interested in.

A comparison problem is about measuring the relationship between two observations. At the word level, this could be asking whether two nouns like *cat* and *dog* belong to the same semantic domain. At the sentence level, this could be asking whether two tweets have a similar sentiment. At the document level, this could be asking whether two articles are examples of a similar style. These questions can be approached using a **text similarity model**. This is a type of unsupervised machine learning in which we as linguists only control the representations being used, not the set of labels used for annotation.

How well do computational models compare with human introspection? In some cases, models can reproduce human intuitions with a high degree of accuracy. For example, text classifiers have been shown to make very good predictions about the part of speech of individual words when trained on small amounts of annotated data. In a case like this, a small amount of seed data, which is annotated by a linguist, supports the analysis of corpora too large to be annotated by a linguist. So a text classifier allows us to scale up introspection-based annotations.

In other cases, computational models can detect patterns in language that are not visible to human introspection. For example, research in both authorship analysis and dialect identification has shown that there are enough individual-specific and community-specific variants to enable accurate predictions of who produced a specific document. But, as linguists, our own introspections are not precise enough to identify these same patterns. In a case like this, computational linguistic analysis makes it possible to answer new questions about language.

Finally, there are cases where computational models completely miss something that is easily accessible to humans. For example, we will follow a case study on multilingualism online which shows that 90 percent of digital language data (from the web and social media) represents just twenty languages. Most languages in the world are low-resource languages from a computational perspective. As a result, many of the computational methods that we cover in this Element are difficult to apply to these languages. As linguists, however, we do not require millions or billions of words in a language before we can begin our analysis.

We need computational linguistic analysis for two reasons: for **reproducibility** and for **scalability**. First, every step in a computational pipeline is fully automated, which means that it can be reproduced and verified. For example, this Element follows five separate case studies that we will introduce in the next subsection. All the graphs and figures and experiments in the Element can be reproduced using the code notebooks that are linked within each section.[1] This is an example of how computational methods support reproducibility.

Second, the once-revolutionary Brown Corpus contained 1 million words (Francis & Kucera, 1967). But it is common now for corpora to range from 1 billion words, like the GeoWAC family of corpora (Dunn & Adams, 2020), up to 400 billion words, like the Corpus of Global Language Use (Dunn, 2020). These very large corpora are often drawn from digital sources like the web, social media, Wikipedia, and news articles. While these sources of language data have tremendous potential for testing linguistic hypotheses on a

[1] And at https://github.com/jonathandunn/corpus_analysis

large scale, working with them requires computational methods to scale up the analysis.

At the same time, the combination of large digital corpora and computational linguistic analysis creates new ethical issues. Given that these corpora contain data from large numbers of individuals, how do we maintain privacy? How do we determine ownership and control over both the data and the models that are derived from the data? How do we prevent models from perpetuating negative stereotypes that are contained in these corpora? We will consider a range of ethical questions like these as we cover the core computational methods.

This first section introduces the basic ideas behind both text classification models and text similarity models. But, before we look at the models themselves, we start by introducing five case studies that we will be following to show how these methods can be used for meaningful linguistic analysis.

1.2 The Case Studies

This Element uses case studies and interactive code notebooks to show you how to apply computational methods using Python as part of a meaningful linguistic analysis. This section introduces the case studies and, at the end of it, you will find a link to a code notebook that introduces the corpora we will be using. Every example, every result, every graph that we use is reproducible given these code notebooks. This availability of both the code and the data is an important part of best practices.

Corpus-Based Sociolinguistics. This case study takes a computational approach to social variation. More precisely, we model geographic variation using digital corpora. These examples use data from the web and social media to model lexical and grammatical variation across different cities and countries (Dunn, 2020). The goal is to find specific linguistic features that are in variation across different populations, as well as to evaluate the distinctiveness or uniqueness of each set of variants. This case study is corpus-based because variants are discovered in naturally occurring corpora rather than elicited through survey-based methods.

Corpus Stylistics. This case study takes a computational approach to forensic linguistics. Do different authors have a predictable style? We use published books from Project Gutenberg (Gerlach & Font-Clos, 2020) to model how authors maintain a unique style across multiple works. What are the best features to capture stylistic variation? How unique are specific authors? Are these authorship models robust or do they depend on a small number of highly predictive features? The goal of this case study is to look at individual variation from a computational perspective.

Usage-Based Grammar. This case study shows how computational models can be used to analyze syntax and semantics. How can we identify which part of speech a word belongs to? How do we find which phrases are collocations in order to treat them as a single unit? Can we extract constructions from a corpus? Is it possible to cluster open-class words into semantic domains by observing their patterns of usage? These examples are drawn from the other corpora as well as from the Universal Dependencies data (Zeman et al., 2021). The goal is to undertake syntactic and semantic analysis while using corpus-based observations instead of introspection.

Multilingualism Online. This case study uses computational methods to analyze underrepresented languages in digital environments. What languages are found online? Where are these languages being used and for what purposes? Which languages have sufficient data and resources to enable a computational linguistic analysis? Can we use computational methods on languages other than English? What are the relationships between different digital registers? In addition to language-mapping data from the earthLings.io project,[2] this case study uses data from Wikipedia,[3] social media, and the web to look at register variation across dozens of languages.

Socioeconomic Indicators. This case study provides examples of how computational methods can be used to answer questions outside of linguistics. For example, how do political and social issues change over time? These examples use data from newspaper articles (Parsons, 2019) and congressional speeches (Gentzkow, Shapiro, & Taddy, 2018) to examine political discourse from 1931 to 2016. This case study further works with customer sentiment in the text of hotel reviews (Li, 2012; McKenzie & Adams, 2018). The goal is to augment traditional survey-based research methods by analyzing large corpora. These questions are not a part of the traditional domain of linguistics. They instead represent new practical applications of corpus analysis.

Taken together, these case studies show how to apply computational methods to a range of problems from different areas of linguistics, using Python to undertake the analysis. You can follow these case studies using the provided code notebooks as well as the associated Python package *text_analytics*.[4] For example, you can use this package to take a closer look at implementation details or to carry out your own analysis of your own corpora. Check out Lab 1.2 to explore the corpora from our case studies, shown in Table 1. After opening the lab capsule through Code Ocean, you will need to run the environment by

[2] www.earthLings.io
[3] www.tensorflow.org/datasets/catalog/wikipedia
[4] https://github.com/jonathandunn/text_analytics

Table 1 List of primary corpora used for the case studies

Corpus Source	Labels	N. Words
Congressional Speeches, 1931–2016	Year, Party	841 million
NYT Lead Paragraphs, 1931–2016	Year	364 million
Project Gutenberg Books	Author	1.04 billion
Tweets, Web Pages	City or Country	836 million
Hotel Reviews	Rating	353 million

following the *Jupyter* link. The notebooks are then contained within the *code* folder.[5]

The remainder of this first section provides an overview of the main topics we will cover. Section 1.3 discusses categorization problems (like part-of-speech tagging) and Section 1.4 discusses comparison problems (like corpus similarity). Section 1.5 introduces one of the central ideas in computational linguistics, that we represent language in a high-dimensional vector space. Finally, Section 1.6 is our first discussion of the ethical implications of computational methods, beginning with the idea of data rights. Each of the four main sections will have a similar structure, ending with a discussion of the ethical implications created by the methods we have just presented.

1.3 Categorization Problems

The first kind of model that we will cover is a text classifier, which we use to solve categorization problems. We start this kind of analysis by deciding which categories are important. In other words, we create a complete classification system, in which each unit of language belongs to one or another category. For example, if we want to apply part-of-speech tags to a corpus, we need to start by defining all the word classes that are available.

Let's say we want to sort tweets by language, in order to build a corpus of social media texts. We first come up with examples of all the languages we are interested in. Some categories might be quite large (a majority class, like English) while others are quite small (a minority class, like Samoan).

Then we train a classifier to automate the labeling task. Labeling here means assigning each text to the correct category. If a tweet is written in Samoan, we want the classifier to label it as Samoan. The goal, of course, is to automate labeling so that we can analyze our categories across corpora containing

[5] Lab 1.2 –> https://doi.org/10.24433/CO.3402613.v1

millions or billions of words. Training here means that we show the classifier examples with their correct labels until the model is able to make accurate predictions on its own.

Let's break down the problem of text classification. First, we need to consider the span of language that we are analyzing. In this Element we will look at examples of classifying individual words (like parts of speech), entire documents (like news articles), and collections of documents (like different writings from a single person).

Second, we need to design a category system. Sometimes this category system is straightforward: for example, if we want to classify documents according to their language or dialect, those categories are already well established (for example, English as a language or New Zealand English as a dialect). But there are other cases where we need to invent a new category system. Let's say we want to classify news articles by topic: We might start with a few high-level topics like SPORTS or POLITICS. But after some experimentation we will most likely find other topics that we have overlooked.

Third, we need to choose our representation to focus on a particular part of the linguistic signal. If the goal is to classify parts of speech, then we care about the surrounding context, especially surrounding function words. But if the goal is to classify news articles by topic, then key terms are more important than local syntactic contexts. And function words will not be helpful for making predictions about the topic of a document. In this Element we introduce four types of representation that allow us to capture different parts of the linguistic signal.

Fourth, we need to train and then evaluate a classifier. Our basic approach is to divide a corpus into **development**, **training**, and **testing** sets so that we can evaluate the model's output labels on samples that it has not seen. This prediction-based evaluation is important for ensuring that the results are valid. This is especially true when the ultimate goal is to use the predictions themselves for further linguistic analysis.

Let's think for a moment about the different kinds of categories that we might be interested in for corpus analysis. These examples come from the case studies. Sometimes we have syntactic categories, like parts of speech: Is this word a noun or a verb? Other times we have semantic categories, like topic: Is this article about SPORTS or POLITICS? The sentiment of a document can be seen as a pragmatic category: Is this review implying a good or bad experience at a hotel? And, finally, sociolinguistic categories involve stylistics or authorship analysis: What dialect does this document represent?

The power of a text classifier is that it allows us to undertake annotation for very different types of linguistic analysis. But we need to make sure that

our representations of language (i.e., our features) correspond with the kind of categories we are working with. And, just as important, we need to make sure our category systems are coherent. Most classifiers are exhaustive and discrete, which means that every sample needs to be a member of one and only one category. For example, an article cannot be about both SPORTS and POLITICS in this framework.

At its core, we as linguists define the classification problem by deciding in advance what the categories will be. Some category systems are scientifically valid: For example, we know that we can identify the dialect or native language of a document's author. But other category systems are not valid: For example, we could not know what social clubs the author belongs to or what their favorite food is. We must establish a good justification for the categories we propose because the classifier will simply replicate any bias that we create in our annotations.

Check out Lab 1.3 to visualize the category systems for some of the categorization problems we will be working with.[6]

1.4 Comparison Problems

The second family of methods that we will cover is text similarity models, which we use to solve comparison problems. The basic idea is to measure how similar two words or two texts are, and then use that similarity to cluster them into groups. Similarity models are not discrete like classifiers and they do not require annotations in advance. For example, let's say we can measure that Charles Dickens writes more like Anthony Trollope than Ernest Hemingway does. Then we undertake that analysis for every pair of writers in a corpus. Now we have a network of relationships between authors that we can cluster into groups of similar authors. The final output is similar to a text classifier (these clusters are categories), except that we as linguists have not defined the labels.

We need to start by thinking about the same questions we posed for designing a text classifier: What span of language are we analyzing (words, sentences, documents) and what part of the linguistic signal are we interested in? For example, we will use similarity models to measure the association between words using both association measures (such as Pointwise Mutual Information) and word embeddings (such as the Skip-Grams with Negative Sampling architecture in WORD2VEC). In this case, the question is about the similarity of a particular word form across an entire corpus. But we will also look at models

[6] Lab 1.3 –> https://doi.org/10.24433/CO.3402613.v1

of corpus similarity and document similarity, which work across much larger spans.

Document similarity, for example, is a method that would allow us to sort news articles into finer-grained categories than a text classifier would support. And, more importantly, we as linguists would not need to predefine an exhaustive set of possible topics. The challenge, of course, is that there is not always an explicit connection between specific terms in an article (*home brew*) and the topic (BEER). So our text similarity model needs to learn that there is a topic in the background that can show up across various terms like *fermenter* and *siphon* and *yeast*. You might search for an article about *how to start a home brew* when you actually need to find an article about *how to soak your grains*. From a linguistic perspective, this is a challenge of finding relationships within a larger semantic domain.

Part of text similarity is the relationship between two texts or two words in isolation. That means that we just compare selections from Dickens and Trollope on their own, without considering other nineteenth-century novelists. But sometimes we want to know the relationship between all the texts in a corpus: The whole web for a search engine, or all English novels for a study of authorship. When we do this, we need a single fixed point for comparison. For example, we could define the location of every city in the world using its angle and distance from Rome and its altitude. Then we use these three numbers to represent where each city is located. Some similarity models work in just this way: We pick a set of points in vector space and map our texts relative to those points. In other words, if every document is represented using the same features, using a table with the same columns, we can directly compare those documents. **Vector space** is a way of thinking about high-dimensional representations of language, an idea that we will look at more closely in the next section. Relationships in vector space should mirror the linguistic relationships we are interested in.

Texts can be similar to one another because they have similar structures (syntax), similar content (semantics), similar implications or sentiment (pragmatics), or because they represent similar authors (sociolinguistics). We all use search engines like DuckDuckGo that work on content-based similarity. But the idea of a comparison problem more generally is that we could also build a search engine that includes authorship (news articles by Canadian women) or sentiment (news articles that have a positive view of urban cycling). The point is that if we can represent a particular part of the linguistic signal, then we can measure similarity between different samples (where a *sample* can be either a word or a document or a corpus).

A similarity model can be used to cluster documents, but it can also be used to cluster words. In addition to this, think about a document as a sample that represents some underlying population: Corpora from New Zealand also represent users of New Zealand English. Thus, we can also use similarity models to cluster abstract objects like dialects: What varieties of English are the most similar, given documents from each variety? Later we will see how to work at all three levels (words, documents, corpora). These problems are actually related to one another: for example, we could start with word similarity to see that *home brew* and *beer* and *keg* and *yeast* are all words that occur in a single semantic script. Then, a model of document similarity would start out knowing which words are related.

Text similarity models do not require us as linguists to define discrete categories in advance. On the one hand, this means it is possible for the model to find categories that we have not considered. We might think of HOME BREW as a topic that includes *beer* and *yeast* but forget about *bottles* and *sanitization*. An unsupervised model is easier to get started with, because there is not the initial work of creating a category system. On the other hand, it is much harder in practice to get these models to work well. The basic problem is that, because we do not tell the model what we want to get as output, we might not like the output that we end up with. Check out Lab 1.4 to further explore the corpora for our case studies.[7]

1.5 Language in Vector Space

The idea behind vector space is that we can find a representation for language in which the relationship between vectors mirrors the linguistic relationships that we are interested in. For example, the vector representation for nouns like *cat* and *dog* should capture the many lexical semantic properties of those words. The first step is to convert language into numeric representations, **vectors**, before we input those vectors into either text classifiers or text similarity models.

The way we choose to vectorize language depends on what part of the linguistic signal we want to analyze. The simplest method is to represent words using their frequency. Let's see what this looks like, starting with the sentences in (1) to (3).

(1) My neighbor sang a song about tulips.
(2) My neighbor sang a song about my neighbor.
(3) My neighbor sang a song about my tulips.

[7] Lab 1.4 –> https://doi.org/10.24433/CO.3402613.v1

Our first step is to make a table, where each word is a column (up and down) and each sentence is a row (left to right). The vocabulary looks like this when we go in alphabetic order:

	a	about	my	neighbor	sang	song	tulips

Now we count how many times each word occurs in each sentence. The number in each cell thus represents the frequency of the word shown in the column header. This means that words are columns and sentences are rows. The first sentence has the word *neighbor* just one time; but the second sentence has it twice. This gives us a frequency vector for each sentence.

	a	about	my	neighbor	sang	song	tulips
(1)	1	1	1	1	1	1	1
(2)	1	1	2	2	1	1	0
(3)	1	1	2	1	1	1	1

This is a toy example because the vocabulary is quite small (only seven words). For actual models we might work with 10k to 50k words. This means that the vector has 10k to 50k dimensions or columns, because each word in the vocabulary has its own column. The larger our corpus, the larger our vector space becomes.

Now let's walk through the process step by step. Take a sentence like (4), shown in the table below. The columns are individual words and the sentence is a row. This is called a **one-hot encoding** because each vocabulary item has a fixed position in the vector. The number in each cell is again frequency.

(4) That hotel has great views of Paris.

	every	great	has	hotel	in	of	Paris	that	views
(4)	0	1	1	1	0	1	1	1	1

Now let's say we have two other sentences, in (5) and (6). These are shown below, together with the frequency vector for (4). We notice a few things here: First, we could measure the distance between these vectors using metrics like Euclidean distance or cosine distance (which we will do in Sections 3.3 and 3.5). Second, small changes in the vectors actually represent large changes in the semantics: (5) is about just one hotel but (6) is about every hotel, a much broader scope for the statement. You will notice that this kind of representation would not work well for capturing scope.

(5) That hotel in Paris has great views.
(6) Every hotel in Paris has great views.