NLP and Word Vectorization

(https://github.com/learn-co-curriculum/dsc-nlp-and-word-vectorization)
(https://github.com/learn-co-curriculum/dsc-nlp-and-word-vectorization/issues/new)

Introduction

In this lesson, we'll learn about some foundational concepts in Natural Language Processing such as stemming and lemmatization, as well as various strategies for converting text data into word vectors!

Objectives

You will be able to:

- Explain stemming and lemmatization
- Explain what stop words are and why they are frequently removed
- Define tokenization in the context of NLP
- Define TF-IDF vectorization and its components
- Define count vectorization and its relationship to bag of words

What is Natural Language Processing?

Natural Language Processing, or ***NLP***, is the study of how computers can interact with humans through the use of human language. Although this is a field that is quite important to Data Scientists, it does not belong to Data Science alone. NLP has been around for quite a while, and sits at the intersection of *Computer Science*, *Artificial Intelligence*, *Linguistics*, and *Information Theory*. In the early days of NLP, it mainly consisted of trying to program algorithms that contained many rules borrowed from the field of linguistics. However, in the 1980s, machine learning started to show great success with many NLP tasks, and many of these rule-based methods took a back seat to approaches involving machine learning and AI. Fast forward to now, and NLP has become an area of applied machine learning that Data Scientists all around the globe work in every day.

NLP and Bayesian Statistics

As machine learning has come into its own, we've seen NLP products get better and better. For instance, in just a few decades, we've gone from rule-based chat bots with preprogrammed responses to things like Siri and
Google Duplex (https://www.youtube.com/watch?v=D5VN56jQMWM)

(https://www.youtube.com/watch?v=D5VN56jQMWM)
(if you aren't familiar with Duplex, take a few minutes to follow that link and watch the demo on YouTube -- you won't be disappointed!). Much of the most exciting advancements currently happening in the field of NLP are due to Deep Learning. However, we can still do amazing things with machine learning and text data by making use of Bayesian methods. For instance, you may remember a time in the early 2000s when the problem of email spam was bad, and getting worse. This problem was eventually solved through the application of machine learning -- specifically, ***Naive Bayesian Classification***! For the remainder of this section, we'll focus on how we can apply our newfound knowledge of Bayesian methods to solve real-world NLP tasks such as **spam filtering** (http://www.paulgraham.com/spam.html) and text classification.

Working With Text Data

Working with text data comes with a unique set of problems and solutions that other types of datasets don't have. Often, text data requires more cleaning and preprocessing than normal data, in order to get it into a format where we can use statistical methods or machine learning to work with it. Let's explore some of the things we generally need to do to get text data into a form where we can work with it.

Creating a Bag of Words

The most common approach to working with text is to vectorize it by creating a ***Bag of Words***. In this case, the name "Bag of Words" is quite descriptive of the final product -- the bag contains information about all the important words in the text individually, but not in any particular order. It's as if we take every word in a ***Corpus*** and throw them into a bag. With a large enough corpus, we'll often see certain patterns start to emerge -- for instance, a bag of words made out of Shakespeare's *Hamlet* is probably more similar to a bag of words made out of *Macbeth* than it is to something like *The Hunger Games*. The simplest way to create a bag of words is to just count how many times each unique word is used in a given corpus. If we have a number for every word, then we have a way to treat each bag as a ***vector***, which opens up all kinds of machine learning tools for use.

Let's explore some of the steps that must occur before we can fully vectorize a text and work with it.

Basic Cleaning and Tokenization

One of the most basic problems seen when working with text data is things like punctuation and capitalization. Although counting how many times a word appears in a text sounds straightforward at

first, it can actually be quite complicated at times, and will almost always require some decisions on our part. For instance, consider the following sentence:

> "Apple shareholders have had a great year. Apple's stock price has gone steadily upwards -- Apple even broke a trillion-dollar valuation, continuing the dominance of this tech stock."

If we were to count how many times each word appears in this sentence, we would likely say that "Apple" has a count of three. However, if we wrote a basic Python script to do this, our algorithm would tell us that the word "Apple" only appears twice! To a computer, "Apple" and "Apple's" are different words. Capitalization is also a problem -- "apple" would also be counted as a different word. Similarly, punctuation is also a problem. A basic counting algorithm would see "stock" and "stock." as two completely different words.

First and foremost, cleaning a text dataset usually means removing punctuation, and lowercasing everything. However, this can be tricky, and require decisions on your part based on the text you're working with and your goals -- for instance, whether or not apostrophes should be removed.

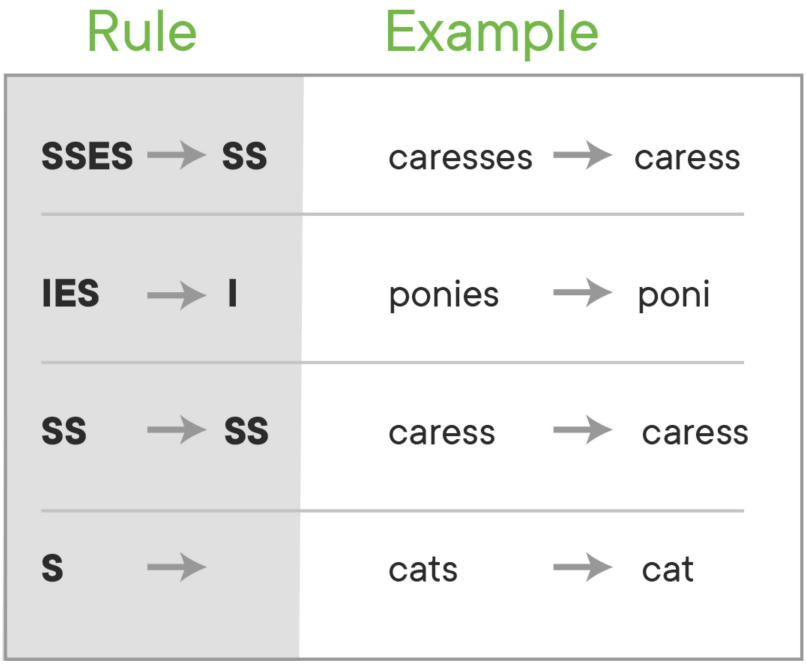
The goal of this step is to create word ***tokens***. The sentence "Where did you get those coconuts?", when cleaned and tokenized, would probably look more like `['where', 'did', 'you', 'get', 'those', 'coconuts']`.

However, there are still other important decisions to make during the tokenization stage. For instance, should "run" and "runs" be counted as the same token, or as different tokens? How about "ran", or "running"?

Stemming, Lemmatization, and Stop Words

Sometimes, depending on the task, it may be best to leave "run" and "runs" as different tokens. However, this often is not the case -- especially with smaller datasets. NLP methods such as ***Stemming*** and ***Lemmatization*** help us deal with this problem, where we reduce each word token down to its root word. For cases such as "run", "runs", "running" and "ran", they are more similar than different -- we may want our algorithm to treat these as the same word, "run".

Stemming accomplishes this by removing the ends of words where the end signals some sort of derivational change to the word. For instance, we know that adding an 's' to the end of a word makes it plural -- a stemming algorithm given the word "cats" would return "cat". Note that stems do not have to make sense as actual English words. For example, "ponies" would be reduced to "poni", not "pony". Stemming is a more crude, heuristic process that contains rule sets that tells the algorithm how to stem each word, and what it should be stemmed to. The process is more crude than lemmatization, but it's also easier to implement. For instance, take a look at this example subset of stemming rules from the **Stanford NLP Group (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html)**:



Lemmatization accomplishes pretty much the same thing as stemming, but does it in a more complex way, by examining the ***morphology*** of words and attempting to reduce each word to its most basic form, or ***lemma***. Note that the results here often end up a bit different than stemming. See the following table for an example of the differences in results:

Word	Stem	Lemma
Studies	Studi	Study
Studying	Study	Study

Finally, you may have intuited that many words in a text are pretty much useless and contain little to no actual information. For instance, words such as "the" and "of". These are called ***Stop Words***, and are often removed after tokenization is complete in order to reduce the dimensionality of each corpus down to only the words that contain important information. Popular NLP frameworks and toolkits such as NLTK contain a list of stop words for most languages, which allow us to easily loop through our tokenized corpus and remove any stop words we find.

Vectorization Strategies

Once we cleaned and tokenized our text data, we can convert it to vectors. However, there are a few different ways we can do this. Depending on our goals and our dataset, some may be more useful than others.

Count Vectorization

One of the most basic, but useful ways of vectorizing text data is to simply count the number of times each word appears in the corpus. If working with a single document, we just create a single vector, where each element in the vector corresponds to the count of a unique word in the document. If

working with multiple documents, we would store everything in a DataFrame, with each column representing a unique word, while each row represents the count vector for a given document.

Document	Aardvark	Apple	...	Zebra
1	0	3	...	1
2	1	2	...	0

Note that we do not need to have a column for every word in the English language -- just a column for each word that shows up in the total vocabulary of our document or documents. If we have multiple documents, we just combine the unique words from each document to get the total dimensionality that allows us to represent each. If a word doesn't show up in a given document, that's fine -- that just means the count is 0 for that row and column.

TF-IDF Vectorization

TF-IDF stands for ***Term Frequency-Inverse Document Frequency***. It is a combination of two individual metrics, which are the TF and IDF, respectively. TF-IDF is used when we have multiple documents. It is based on the idea that rare words contain more information about the content of a document than words that are used many times throughout all the documents. For instance, if we treated every article in a newspaper as a separate document, looking at the amount of times the word "he" or "she" is used probably doesn't tell us much about what that given article is about -- however, the amount of times "touchdown" is used can provide good signal that the article is probably about sports.

Term Frequency is calculated with the following formula:

$$Term\ Frequency(t) = \frac{number\ of\ times\ t\ appears\ in\ a\ document}{total\ number\ of\ terms\ in\ the\ document}$$

Inverse Document Frequency is calculated with the following formula:

$$IDF(t) = log_e(\frac{Total\ Number\ of\ Documents}{Number\ of\ Documents\ with\ t\ in\ it})$$

The ***TF-IDF*** value for a given word in a given document is just found by multiplying the two!

Summary

In this lesson, you learned about some foundational concepts in Natural Language Processing such as stemming and lemmatization, as well as various strategies for converting text data into word vectors.