Assignment 1: Vectorization and NLP

Dakota Murray
Applying ML techniques to CL

February 23, 2019

Chosen task: Citations between scientific publications are one of the most important types of data in bibliometrics, my area of research. However, not all citations are the same. Moreover, it can be difficult to determine, without manual effort, *why* one paper cites another. This is particularly an issue for the bibliometric analysis of data objectsthe usage and re-use of data is increasingly important for scientific research, and citations offer one means to study how data is shared and distributed between publications. However, data does not tend to have its own citation, and instead tends to be associated with publications. Moreover, these publications are rarely *just* about the data, and tend to introduce a method or conduct a study along with providing the data. So how do we determine if a citation to one of these publications is a citation to the study, or a citation for the data? This is the task that I will focus on for this, and potentially future assignments. The task is: classify a citation to an ambiguous citable object (publication that is associated with shared data, as well as a method or a study) using the characteristics of the text surrounding the in-text citation.

For this assignment, I will focus on citations to the paper *The Use of Multiple Measurements in Taxonomic Problems* by R. A. Fisherthis paper introduces the method of Linear Discriminant Analysis, as well as the commonly-used Iris Setosa dataset. The goal is to determine whether a citation to this paper is referencing the method, or the data (or both?)

Task 1

I applied the Stanford coreNLP parser to the following sentence, and my vectoraization strategy is informed by this output:

> "fixed-size LS-SVM classifiers using linear and RBF kernels , as explained above, were compared against k-nearest-neighbours (kNN), linear discriminant analysis (LDA) and random forest . The kNN classification was implemented using the Matlab function fitcknnand the optimal k is obtained with 10-fold CV. LDA was implemented using the fitcdiscr function, whereas the random forest classifier was implemented using TreeBagger with 100 bagged trees"

Below I give a brief run-through of my vectorization strategy, and how and why I would (or wouldn't) use each of these features produced by the Stanford coreNLP parser.

Parts of Speech: One issue with applying NLP techniques to natural language is the incidence of new or unknown words. This is especially true for technical language, where new highly-technical terms proliferate across disciplines. These words tend to be the names of techniques or concepts, like "k-nearest-neighbors" and "TreeBagger" above. These tokens are likely to be useful to determining whether a citation references the method or the data associated with the citing object; however, the variance and sheer quantity of such terms means that rule-based identification may not be practical. I can use parts-of-speech tagging to tag noun phrases like these and extract them as tokens, using their presence as a feature in my vectorization.

Moreover, like all natural language, words in technical parlance can have different meanings when used in different ways. Parts of speech tagging can help resolve this ambiguity. So in a typical, non PoS tagging bag-of-words vectorization, I would include tokens like "classifiers" and "using". However, using PoS tagging (and with a lot of data!) I can instead construct tokens like "classifiers_nns" and "using_vbg", thus allowing for differentiation between usage of certain tokens. However, this would also lead to a sparser and larger feature space, which may make classification more difficult and less interpretableas such, I would only use PoS tagged tokens if there was a non-trivial increase in performance.

Named Entities: Named entities are used a great deal in the processing of scientific publicationsone common case is for the automatic identification of drugs or species in biomedical texts. Here, named-entity recognition might be used to extract tokens relating specifically to names and methods. For example, the coreNLP parser tags the acronym "LDA" as an organizationeven though this is wrong, if the tagger consistently tags methodological acronyms, then it might prove useful to identify and extract specific acronyms as tokens, and identify those most associated with citations of the methods vs. the data.

Additionally, I found that the named entity recognition can identify numbers. I can also extract tokens representing numbersif a number is used that matches the size of the data in the Iris Setosa dataset, then this could indicate that the data being used, rather than the method.

Lemmas: This text is all in past-tense, and so it is fairly morphologically consistent. However, if I extended this analysis to other sentences, then lemmas could prove very useful. Often, the introduction of a publication is present or future tense, whereas the method and results are in past-tense. Additionally, norms of tense vary between scientific discipline and journal. Lemmatization could prove useful in normalizing the morphological variance of words and creating a less-sparse feature space. Therefore, I would lemmatize all tokens before including them into my feature space.

Coreference: All citations have a context, which may span before and after the sentence actually containing the in-text citation. However, extracting all tokens from all sentences in a citations context is impracticalthis would create a large and noisy feature space. Instead, there needs to be a way of determining which sentences to process and to include as features. In this case, we could identify a term of interest: "LDA", and identify that it is mentioned in both the 1st, as well as the 3rd sentence in this text. Therefore, I could vectorize the first and third sentences into the same feature vector, while skipping the second sentence.

Dependency parse tree: I will be honest, I don't really understand dependency parsing. The resulting trees and relationships are far too complicated to be practical in many applications. I have read that dependency feature have proven somewhat useful for classi-

fication of citation function. However, I am not entirely convinced. If I were to use them, I would build a bag-of-words model but in addition to tokens of individual terms, I would also include tokens of dependency structures, like "fixed-size_amod_classifiers".

Constituent Parse Trees: This parse breaks down the sentences into individual phrases. It might prove useful if I wished to ignore parts of the sentence, and for example, vectorize only noun phrases. However, I am not convinced that this would be useful for my purposes, and so I would probably avoid this parser.

In summary: I tend to prefer simpler and smaller feature spacesthese are often easier to interpret and modify for new text, and computationally cheaper when applying to large datasets. I would avoid parts-of-speech tagging, constituency parsing, and dependency parsingthese methods are computationally slow, will lead to a sparser feature space, and I am not convinced that they will lead to noticeable gains for classification. Instead, I would first extract only tokens related to named entities along with their associated tag, such as "LDA_organization" or "Fisher_person". I would also extract lemmatized versions of all terms, while ignoring stop-words. I would extract all tokens from the sentence containing the in-text citation, and from the surrounding two context sentences, if there is a co-reference between one of these context sentences and the citing sentence. The resulting vector would have a value for each token, and would simply be a boolean value indicating presence of the token in the sentence.

Task 2

I ran the code through the spaCY parser, and selected for parts of speech, dependency parsing, lemmatization, shape, and whether or not it is a number. I also parsed for named entities.

The output of spaCY differs from that of the Stanford coreNLP Parser. for one, spaCy does not provide (that I can see) dependency parse trees, though it still provides the dependency structure tag for each token. Also, by default, spaCy does not automatically detect noun phrases; whereas coreNLP identified "k-nearest-neighbors" as a single token, the default spaCy parsing instead treated these as three separate tokens: "k", "nearest", and "neighbors". This in turn affects parts of speech tagging: the term "fixed-size" is an adjective, or could be considered part of a noun phrase; however, in spaCy, "fixed" was identified as a single token and a verb, which is not the case in the sentence.

Named entity recognition was also different. spaCy did not identify "LDA" as a named entity, but it did identify "SVM", "RBF", "Matlab", "CV", and "TreeBagger"however, the entity recognition algorithm did identify "Matlab" as a person, and the other acronyms as organizations. Still, these features should prove useful for classification.

I would mostly follow the same strategy as I devised with the Stanford coreNLP parser. Except now, I cannot rely on noun phrases or other such features. I will first extract named entities and corresponding tags, because I feel that these would be important features for classification tasks. I would then extract unigrams, bigrams, and trigrams from the lemmatized tokens, excluding those n-grams that do appear only rarely or which are not associated with any particular classification. The n-grams will help to capture the multi-word noun phrases that spaCy seems to miss. However, this will also create a very large and

sparse feature spaceas such, I will not tag words with PoS of dependency tags, which would only server to make the feature space even more sparse.

Task 3

The code and output for this task can be found in the python notebook submitted along with this assignment.

‘

‘