An Attempt to Compare the Performance of Word Embedding Algorithms at Clustering English Wikipedia Articles

Michael Wheeler
wheeler.m@northeastern.edu

Abstract

For the final project in *DS5230: Unsupervised Machine Learning and Data Mining*, I attempted to generate a clustering of English Wikipedia articles using state-of-the-art word embedding algorithms in tandem with traditional clustering techniques. I aimed to compare the performance of word2vec and fastText on this task by training each model on the English Wikipedia corpus, then using those trained models to generate an embedding for each article to be used for clustering. Unfortunately I was unable to generate clustering results on which to report; instead I discuss the technical challenges I faced in my attempts, areas for improvement in my methodology, and the potential for future work in this space.

1 Background

Wikipedia is an online, open-source, worldwide, collaborative encyclopedia. Wikipedia allows anyone in the world to contribute content to its pages in real-time, so long as those contributions are accurate, sourced appropriately, and free of both original research and editorial content. Since its founding in 2001 by Jimmy Wales and Larry Sanger it has grown into the largest and most-trafficked reference website on the Internet, with tens of millions of articles written in hundreds of languages and spanning virtually all areas of human knowledge. More specifically the subset of Wikipedia content written in the English language ("English Wikipedia", the focus of this project) currently totals over 6 million articles. (Wikipedia 2020a)

In the field of unsupervised machine learning, clustering is the task of assigning data points to one or more groups based on the similarity of their features. The goal is to produce groups whose members have high similarity between each other and low similarity with respect to members of other groups. What separates clustering from classification is that those data points do not come with any group labels on which an algorithm may train. Thus, a successful clustering algorithm must learn the latent structure of the data well enough to produce meaningful separation given only the unlabeled data itself and the desired number of clusters. Two of the most commonly-used clustering algorithms are k-means clustering and spectral clustering: both techniques are partitional, meaning that all data points are contained in exactly one cluster. K-means clustering is an iterative algorithm in which each point is assigned to a cluster based on its closest centroid; those centroids are initialized at random or heuristically and then recomputed based on its assigned points until convergence. Spectral clustering is an algorithm that improves on k-means with a carefully-constructed design matrix: the original data points form a similarity graph, the adjacency matrix and degree matrix of that similarity graph are combined to form a Laplacian matrix, and finally a subset of the eigenvectors of that Laplacian matrix serve as the input to the standard k-means algorithm.

One major limitation of both these algorithms is that they require a design matrix: where each row vector corresponds to a particular observation in the dataset, and each column vector corresponds to a particular feature across all observations. This format — while suitable for numeric data — poses a challenge for less-structured data like text. Specifically for the task of clustering Wikipedia articles based on their content (the focus of this project), the use of these algorithms requires a method of vectorizing the text of each document to create a representative matrix for clustering. Such methods exist in the form of word embedding algorithms, whose goal is to map the vocabulary of a training corpus into a vector space whose arithmetic mirrors the semantic relationships between the words as observed in the corpus.

Perhaps the most famous technique used for learning word embeddings is word2vec; the approach uses one of two different prediction tasks to train and evaluate its embeddings. A feed-forward neural network accepts a one-hot encoding of words and outputs the learned vector representations of those words based on the weights of the network's hidden layers. Then the network's performance is evaluated by using those vectors to predict a target word given a fixed-size context window of words occurring before and after said target word, across all the text in the training corpus. Alternatively, the vector representations can also be evaluated on the complementary task: by predicting the occurrence of the context words given a particular target word, making use of negative sampling to make sure the model doesn't learn only positive predictions. (Mikolov et al. 2013b; 2013a) Another popular algorithm for word vectorization is fastText; this approach builds on word2vec by constructing n-grams and skip-grams from individual characters instead of words to better represent words with similar prefixes, suffixes, and root phrases. (Bojanowski et al. 2017)

2 Related Work

While preparing for this project I encountered two works with similar approaches, both by the same author, for the purposes of information retrieval and search. (Szymański 2011; Szymański and Dziubich 2017) In both papers the author performs spectral clustering on Wikipedia articles and uses the human-generated article taxonomy included in the WikiMedia markup to evaluate the quality of the clusters externally. In the earlier work, the documents are vectorized using a bag-of-words approach, where the value for each word is its term-frequency inverse-document-frequency score (abbr. TF-IDF score), a common measure of the importance of some to that particular article. (Szymański 2011) The later work uses an identical bag-of-words approach with TF-IDF scores, but augments the document vector representation with one-hot encoding of links to other articles and performs dimensionality reduction on those vectors using principal component analysis. (Szymański and Dziubich 2017)

3 Methodology

Figure 1 outlines the data pipeline I wrote over the course of this project.[1]. The Wikipedia data source file is a bzip2-compressed archive, 18.9GB in size, consisting only of the most recent revisions of all articles belonging to English Wikipedia (i.e. no older revisions, no discussion pages, no user pages, no meta pages, and no pages pertaining to non-English Wikipedia) as of 2020-11-01. The archive, along with its index for decompression, was downloaded from a WikiMedia dumps mirror. (Wikipedia 2020b) I used the WikiExtractor Python library to decompress the archive: the tool discards most of the MediaWiki markup in favor of retaining the text, and splits up the articles into text files containing XML-like tags for each article. (Attardi 2020) WikiExtractor took approximately six hours to run to completion and produced over 14,000 text files totaling 14GB in size.

In the next step I wrote a Luigi task to extract the text of each article from the malformed XML files, generating both a single large corpus file and a Dask Dataframe with 14,000 partitions. Luigi is a framework for orchestrating repeatable data science workflows and pipelines, and is used to structure the rest of the pipeline downstream from WikiExtractor. (Luigi 2020) Dask is an extension of Numpy and Pandas for handling larger-than-memory datasets: it is designed around the delayed computation of task graphs, and comes with a sister library Dask-ML used in the clustering task downstream. (Dask 2020) After extracting each article's text from the XML but before writing to disk I performed tokenization on the article text using NLTK — a library providing utilities for a variety of natural language processing tasks — in preparation for use with the word embedding algorithms. (NLTK 2020)

Next, I trained the fastText and word2vec models on the tokenized corpus file generated in the previous task, using implementations for each algorithm provided by the library Gensim. (gensim 2020) For both models I trained for three

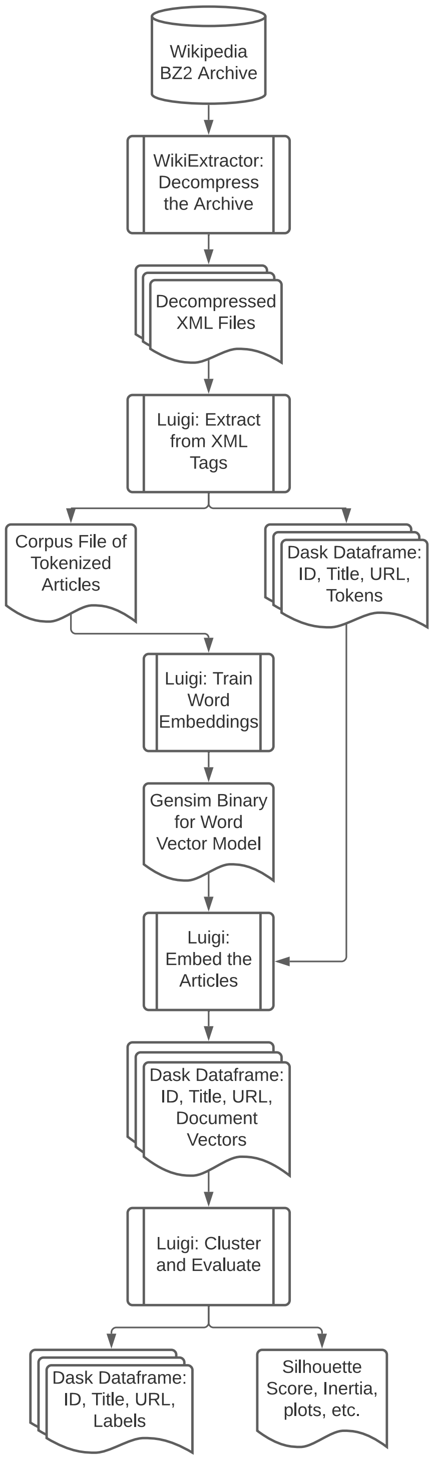
Wikipedia BZ2 Archive
WikiExtractor: Decompress the Archive
Decompressed XML Files
Luigi: Extract from XML Tags
Corpus File of Tokenized Articles
Dask Dataframe: ID, Title, URL, Tokens
Luigi: Train Word Embeddings
Gensim Binary for Word Vector Model
Luigi: Embed the Articles
Dask Dataframe: ID, Title, URL, Document Vectors
Luigi: Cluster and Evaluate
Dask Dataframe: ID, Title, URL, Labels
Silhouette Score, Inertia, plots, etc.

Figure 1: A flowchart of the model preprocessing and training pipeline generated and executed for this project.

[1]Code available at https://github.com/mikewheel/spectral_clustering_word_embeddings

epochs with a window size of four words and a minimum frequency of ten occurrences in the corpus to generate word vectors of size 200. The resulting binary model files were each about 45MB in size. I then used those trained word embedding models to embed each article: as a simple strategy I concatenated the element-wise minimum and element-wise maximum vectors, each calculated across all the word vectors in the article, for a document embedding of size 400. (StackExchange 2016)

Finally I attempted to perform both k-means and spectral clustering on the document embeddings for the entirety of the English Wikipedia corpus. The goal was to produce both trained clustering models, whose attribute for inertia (SSE) could be extracted via inspection; as well as a Dask Series of cluster labels for each article, from which the silhouette score could be calculated using scikit-learn's implementation of that particular metric. (Dask-ML 2020; scikit learn 2020) The particulars of what I attempted and where they failed are discussed in greater detail in Section 4.

All of the above computation was performed on a single Elastic Cloud Compute (EC2) instance available on Amazon Web Services (AWS). The instance type I used was t2.2xlarge: with 8 cores and 32GB of RAM, along with an Elastic Block Store (EBS) for storing data and model assets totaling 120GB in size.

4 Results and Challenges

Despite my best efforts I could not get any clustering to complete in time for the project deadline. This is due to a number of factors, mostly related to the scale of the data once I reached the clustering step in the pipeline. First, one of the advantages of the Dask Dataframe during the previous steps is that it allows for efficient computation and use of memory by lazy evaluation over its many partitions — in this case, over 14,000. However, the clustering implementations provided by Dask-ML (k-means clustering and spectral clustering) would not accept a Dataframe, but instead required that the input be converted to a Dask Array (the Dask analogue of Numpy's ndarray class). In order to perform that conversion the entire Dataframe containing all of the tokenized Wikipedia articles had to be loaded into memory: a step that on its own took 45 minutes on each attempt.

What is more, upon performing this conversion I discovered a bug with regards to reading my recently saved Dataframe from its Parquet files on disk — there was no help on Stack Overflow or anywhere else online. In fact, the errors I received point to the possibility that I discovered a bug in Dask itself; as a result I created an issue on GitHub to alert the library's maintainers.[2] This setback forced me to not only spend time debugging in vain, but also to re-run the upstream tasks in order to switch over to the much-slower CSV format at that point in the pipeline.

Then, once the previous data assets were re-generated and loaded into memory for the appropriate conversion, I encountered an overflow/not-a-number error that arose somewhere during the process of Laplacian generation or eigenvector calculation in the spectral clustering process. Due to the lazy-evaluation of Dask's larger-than-memory data types I was unable to use PyCharm's debugger to inspect the intermediate steps in the same way that I could for a pandas Dataframe. I did however run a check on the document embeddings for infinite values and not-a-number data points and confirmed that there was no invalid data in the input, meaning the erroneous data came about somewhere within the black-box of the spectral clustering implementation. Finally I decided to abandon spectral clustering, and turned to Dask-ML's implementation of k-means clustering instead. That particular model — configured to generate only two clusters with a maximum of two iterations for expectation maximization — did not error out, but rather ran for sixteen hours without terminating or producing any results, at which point I was forced to abandon the effort.

It is worth noting that at an earlier stage in the project I had considered sub-sampling the Wikipedia corpus to verify the pipeline from end-to-end on a smaller dataset and produce initial results. I decided against doing that for two reasons. First, because the semantic content of Wikipedia is so diverse and the curse of dimensionality is especially pronounced in 400 dimensions, I was concerned that the clusters produced on a smaller randomly-selected sub-sample wouldn't hold much meaning. I didn't want to invest the time in coming up with an intelligent and representative sub-sampling strategy when I had other problems to tackle for the project. Second, up until the final clustering step I hadn't encountered any problems with either the data format or the running time — including while training the word embeddings on the full corpus. Given the smooth sailing on upstream tasks, I thought that the clustering would not perform dramatically slower than any of the previous steps in the pipeline. Obviously this was a critical error and a poor judgment call on my part.

5 Discussion

Working with an in-memory dataset first to discover and correct errors in end-to-end performance is a critical part of developing pipelines to process larger-than-memory datasets, and a lesson hard-learned during the course of this project. With that in mind, I believe that the development effort of this project was not in vain. The pipeline I have created is set up such that with more time and development effort, a clustering on the full Wikipedia corpus could be produced and evaluated with the desired metrics described in Section 3.

Given more time to work on this project in the future, I believe that the highest priority is more intelligent experimentation with the hyperparameters at each stage: both during the training of the word embeddings and during the clustering itself. For instance, I would be interested in training fastText and word2vec for a greater number of epochs and with better tuning of currently-default parameters such as context window size and evaluation task. Of particular interest to me is the tradeoff between word embedding size and clustering performance: a smaller embedding size would allow the clustering to proceed more quickly, but at what cost to the quality of the results?

[2]https://github.com/dask/dask/issues/6964

In a similar vein, I would also prioritize the evaluation of the word embeddings on tasks similar to those presented in their original papers before using them to embed the documents. Within the time frame of this project I made no effort to evaluate the quality of the word embeddings, leaving me with little to no idea as to how much signal is captured when they are applied to the articles. In my opinion this is a major flaw in my methodology that could render any downstream results meaningless. I would also like to explore a less naive strategy for performing the document embedding: training an additional sequence-to-sequence model such as a Transformer would likely translate to a better representation on which the clustering can learn.

Lastly, I would spend more time considering exactly what I hoped to learn from the clustering and how I expected them to look in a qualitative way. I chose the project out of broad curiosity about Wikipedia and the diversity of its data; however a more focused approach with clearer objectives about the specific insights I could gain from clustering would likely prove worthwhile. At the time that I proposed the project, the class had not yet covered community detection techniques. In hindsight, I think that investigating community detection across Wikipedia articles might have been a more fruitful endeavor. As the related literature suggests, the inherent link structure of Wikipedia articles provides a great deal of information that the text alone does not. Leveraging this information may produce groupings that are more semantically meaningful than choosing an arbitrary number of clusters across many millions of very diverse articles based on their embedded text content alone. What is more, a smaller data representation in the form of a graph alone means that training and evaluation could likely occur much faster — no need to process text at all.

At the same time clustering on Wikipedia could also prove useful beyond exploring Wikipedia itself. Prof. Yehoshua suggested (along with the related literature) that the taxonomic structure of Wikipedia could be used to evaluate how well the clustering captures semantic meaning from the text. It is conceivable that using the existing code as a proof-of-concept, I could design a robust and diverse benchmark for evaluating novel document clustering algorithms around the Wikipedia corpus and all its associated metadata. By leveraging the many layers of taxonomy and hierarchy generated by Wikipedia's contributors, an "ideal" semantic clustering of arbitrary size could be extracted from the dataset and used to objectively evaluate similarity between the groups produced by contributors and those produced by a clustering algorithm or novel vectorization technique for documents.

References

Attardi, G. 2020. attardi/wikiextractor. original-date: 2015-03-22T12:03:01Z.

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching Word Vectors with Subword Information. *arXiv:1607.04606 [cs]*. arXiv: 1607.04606.

Dask-ML. 2020. dask-ml 1.7.1 documentation - Clustering.

Dask. 2020. dask/dask. original-date: 2015-01-04T18:50:00Z.

gensim. 2020. RaRe-Technologies/gensim. original-date: 2011-02-10T07:43:04Z.

Luigi. 2020. spotify/luigi. original-date: 2012-09-20T15:06:38Z.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. arXiv: 1301.3781.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013b. Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546 [cs, stat]*. arXiv: 1310.4546.

NLTK. 2020. nltk/nltk. original-date: 2009-09-07T10:53:58Z.

scikit learn. 2020. sklearn.metrics.silhouette_score — scikit-learn 0.23.2 documentation.

StackExchange. 2016. Cross Validated: Apply word embeddings to entire document, to get a feature vector.

Szymański, J., and Dziubich, T. 2017. Spectral Clustering Wikipedia Keyword-Based Search Results. *Frontiers in Robotics and AI* 3. Publisher: Frontiers.

Szymański, J. 2011. Categorization of Wikipedia Articles with Spectral Clustering. In Yin, H.; Wang, W.; and Rayward-Smith, V., eds., *Intelligent Data Engineering and Automated Learning - IDEAL 2011*, Lecture Notes in Computer Science, 108–115. Berlin, Heidelberg: Springer.

Wikipedia. 2020a. Wikipedia: About. Page Version ID: 992972317.

Wikipedia. 2020b. Wikipedia:Database download. Page Version ID: 990954890.