Exploring Word Embeddings

Davis Busteed — LING 581

https://github.com/dbusteed/embedding-explorer

Table of Contents

ABSTRACT

Vector semantics, the process of mapping words and phrases into a mathematical vector, is "the standard way to represent meaning in [natural language processing]" (Lonsdale, 2019). With a vector semantics model, an "embedding" is given to each word or phrase in a document. Word embeddings are a crucial input for many NLP applications, but as with many topics in computational linguistics, simply understanding the algorithm behind vector semantics can only get you so far. To make vector semantics more approachable, I developed a tool called The Embedding Explorer. By creating "model snapshots", users of the tool can observe how word embeddings shift during the formation process.

RESEARCH

Before beginning this project, my understanding of word embeddings was limited to our class discussion on vector semantics (Lonsdale, 2019). As I decided to move forward with The Embedding Explorer, I researched different methods for creating word embeddings as well as methods for visualizing high-dimensional vectors.

TF-IDF, PPMI, and Word2Vec are some of the different approaches to representing semantics via embeddings (Lonsdale, 2019). I decided to use Word2Vec because it produces a "dense" model (rather than a "sparse" model produced with TF-IDF). Although I used the default tuning parameters when implementing the Word2Vec

algorithm, I was aware that "model tuning" could be accomplished by adjusting these parameters (Rong, 2014).

My final area of research focused on the feasibility of using Principal Components Analysis to reduce the dimensionality of the embedding vectors to $p = 2$, so that I would be able to plot the embeddings in 2D space. I had used PCA in different settings, but I also found that PCA has been used in conjunction with word embeddings by other researchers (Lebret & Collobert, 2013).

HYPOTHESIS

As I focused on the primary goal of The Embedding Explorer, I began to hypothesize that the formation of a word embedding could be simulated by creating multiple semantic models, or "snapshots". Each snapshot would use a small section of the available text along with the text used in the previous snapshot. This "cumulative" approach would provide a representation of any given word at its current state. This seemed reasonable, so I decided to implement this theory and evaluate how well it worked.

METHOD

Tool Development

I developed The Embedding Explorer using Python and the PyQt GUI toolkit. After creating the basic interface with PyQt Designer, I wrote the backend code that would

handle user input, create embedding models, and perform the other functions previously identified. I used Gensim's implementation of the Word2Vec algorithm for creating word embedding models. Other modules such as NLTK and Scikit-Learn were used for manipulating text and finding principal components, respectively.

Core Functionality

The following sections outline the process of exploring the formation of word embeddings. This is the core functionality of The Embedding Explorer.

1. Load Lexical Resources

The Embedding Explorer allows the user to manually input text (either by typing or pasting-in text) or uploading a text file. Regular Expressions (**`sub`**) and NLTK functions (**`wordpunct_tokenize`**, **`sent_tokenize`**) were used to format the text into the appropriate structure required for creating the word embeddings. Words were also converted to lowercase, and contractions were replaced with their elongated form during this step. See Listing 1 for an example of the transformation performed to the text.

$$\text{This is an example. It's very simple} \Rightarrow \big[[\text{this, is, an, example}], [\text{it, is, very, simple}]\big] \quad \textbf{(1)}$$

2. Create Word2Vec Models

After preprocessing the text into a suitable format for Word2Vec, sentences were then split into "chunks". Since each chunk would contain the sentences used to create a

model snapshot, the sentences were “chunked” in a cumulative manner. See Listing 2 for an example of the chunking process, in which four sentences are transformed into four “cumulative chunks”.

$$[\, a, b, c, d \,] \Longrightarrow [\, (a), (a, b), (a, b, c), (a, b, c, d) \,] \quad \textbf{(2)}$$

Separate word embedding models (snapshots) were then created from each of these chunks. Aside from setting **`min_count`** equal to 1, default parameters were used during the initialization of each **`Word2Vec`** object.

3. Visualize Model Snapshots

With several model snapshots created, users could now select a word within the vocabulary, and visualize the formation of the word embedding. To facilitate this, The Embedding Explorer would plot the 10 most similar words to the selected word for each model snapshot, and let the user view each model’s set of similar words.

Similar words were found using the **`similar_by_word`** method attached to the **`Word2Vec`** object. After which, the first two Principal Components were calculated for each word embedding. By reducing the dimensionality of the embeddings to $p = 2$, words could be plotted easily be plotted (and compared) on a 2D plane.

Other Features

After implementing the core functionalities of The Embedding Explorer, I decided to "spread the fixed costs" of my development time by adding some additional features to the tool.

1. Basic Model Information

Because one of the goals of The Embedding Explorer is to bridge the gap between abstract vector semantics and real-word understanding, I chose to display the vector of floating-point integers of the selected word. Other basic model details are displayed on The Embedding Explorer's main interface.

2. Similarity Chart and Table

Although similar words can be seen when viewing the final model snapshot, I chose to extend this feature by allowing users to view similar words in both a chart and table format. Because the charts are made with Matplotlib, users can view the charts in the Matplotlib viewer, which allows them to zoom in, reshape axes, and save figures to their device.

3. Random Charts

I extended the functionality of the similarity charts and developed a way for users to visualize relationships among random words. This isn't necessarily useful for specific

analysis, but it still fulfills the overall goal of making word embeddings more accessible by giving users a chance to interpret a random subset of word embeddings.

4. Embedding Arithmetic

Although it wasn't in my original project definition, I decided to make it possible for users to do "embedding arithmetic", in which words can be added and subtracted using the corresponding embeddings.

5. Import / Export Models

Lastly, I implemented features for importing and exporting the Word2Vec models. This simple feature can go a long way by allowing users who are familiar with vector semantics to quickly create embedding models with a GUI tool.

RESULTS

Evaluating the results of The Embedding Explorer was tricky because we don't have a "gold standard" for what word embeddings should look like. Using machine learning terminology, it seems that word embedding is similar to an unsupervised learning approach, in which we can't "double-check" our work.

Although The Embedding Explorer is the only tool (as far as I know) that uses model snapshots to explore the formation of word embeddings, there are other tools that use word embeddings to visualize similarities between words. I experimented with WordVis, a tool that maps the similarity between words in WordNet. WordVis has a "stricter"

definition for what constitutes similarity between words, making it difficult to directly compare with The Embedding Explorer. As seen in Figures 1 and 2, The Embedding Explorer identifies other months as being similar to the month October, but WordVis only highlights alternative spellings and abbreviations of the word.

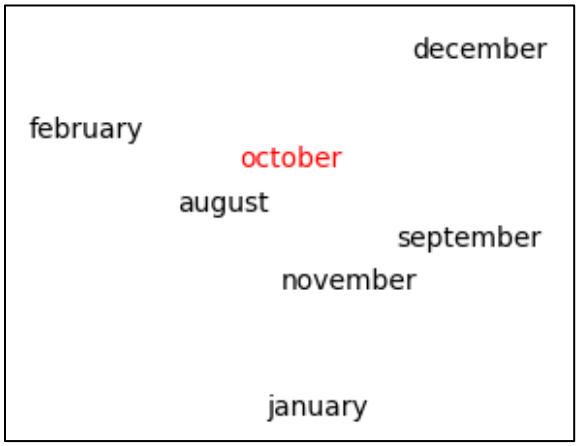

Figure 1 – The Embedding Explorer, similar words to "October"

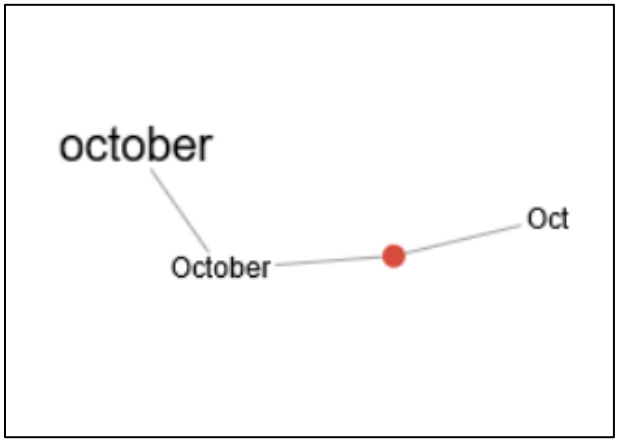

Figure 2 – WordVis, similar words to "October"

Overall, I trust the results provided by The Embedding Explorer. I tested several examples during the development process to ensure that preliminary steps like preprocessing were giving consistent results.

Moving forward, I plan on optimizing the creation of model snapshots and managing exceptions from the GUI in a more organized manner.

BIBLIOGRAPHY

Lebret, R. & Collobert R. (2013). Word Embeddings through Hellinger PCA. https://arxiv.org/abs/1312.5542

Lonsdale, D. (2019). Computational Lexicography. http://linguistics.byu.edu/classes/ling581dl/compulex.pdf

Rong, X. (2014). word2vec Parameter Learning Explained. https://arxiv.org/abs/1411.2738