1 Word Embeddings

For my final project I chose to apply principal component analysis to word embeddings. Word embeddings allow the representation of words as vectors, where words with similar meanings have similar vector representations. One naive algorithm to embed words into a vector space would be to gather the set of all words in the input dataset, and then collect for each word the frequencies of its neighbors across all its occurrences (within some fixed window). For example to embed the word ”math” using this algorithm, we would initialize the embedding of ”math” as an n-dimensional vector where n is the number of unique words in the dataset. We would then look at each occurrence of ”math” in the input dataset, and for the words within some distance of ”math,” we would increment their corresponding count in the embedding. When we finish this process, the resulting vector captures which words in the corpus ”math” occurs most frequently around.

In practice this approach is not commonly used, because there are many unique words in large corpora, and therefore if we were to use this technique the embeddings generated would be extremely high-dimensional. Instead, a popular approach for generating word embeddings is to use Word2Vec, a technique published by Google in 2013. Word2Vec that uses a shallow neural network to embed words into a vector space, usually of some dimension between 100 and 1000. I used the `gensim` library in python to generate word embeddings of dimension 100 for my analysis. [**?**] is a set of macros built atop

2 Dataset

For my dataset I chose to use the Brown Corpus, which is a large compilation of American English, containing roughly one million words. It was compiled at Brown university in 1961, and it contains 500 samples across many different genres. The `nltk` python package provides a simple interface for the Brown Corpus, which I used to generate my word embeddings.

3 PCA And Results

I implemented PCA so I could reduce the dimensionality of the resulting word embeddings from dimension 100 to dimension 2. This allows the word embeddings to be visualized in 2d plots. Here are some example words for which I found the top 10 nearest neighbors in the 2d vector space using cosine similarity, and used my implementation of PCA to reduce the set of 11 100-dimensional vectors to a set of 11 2-dimensional vectors. I then used the `matplotlib` python library to make scatter plots.

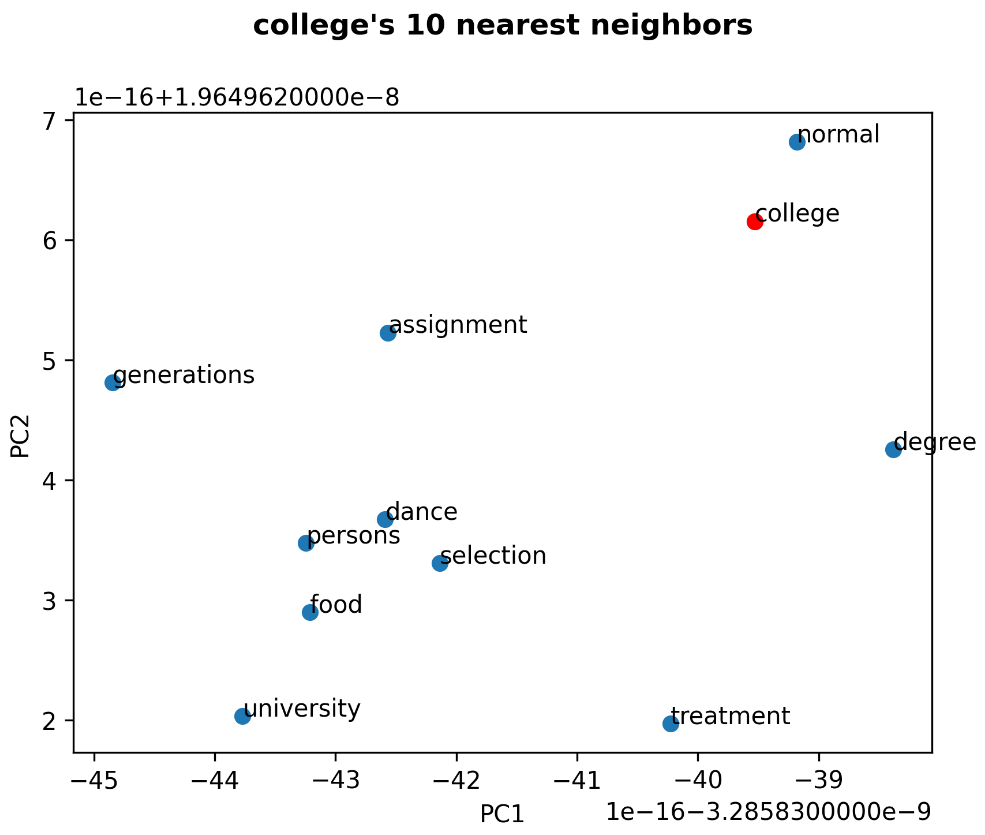

Figure 1: Top 10 nearest neighbors to ”college” in the 100d vector space using cosine similarity

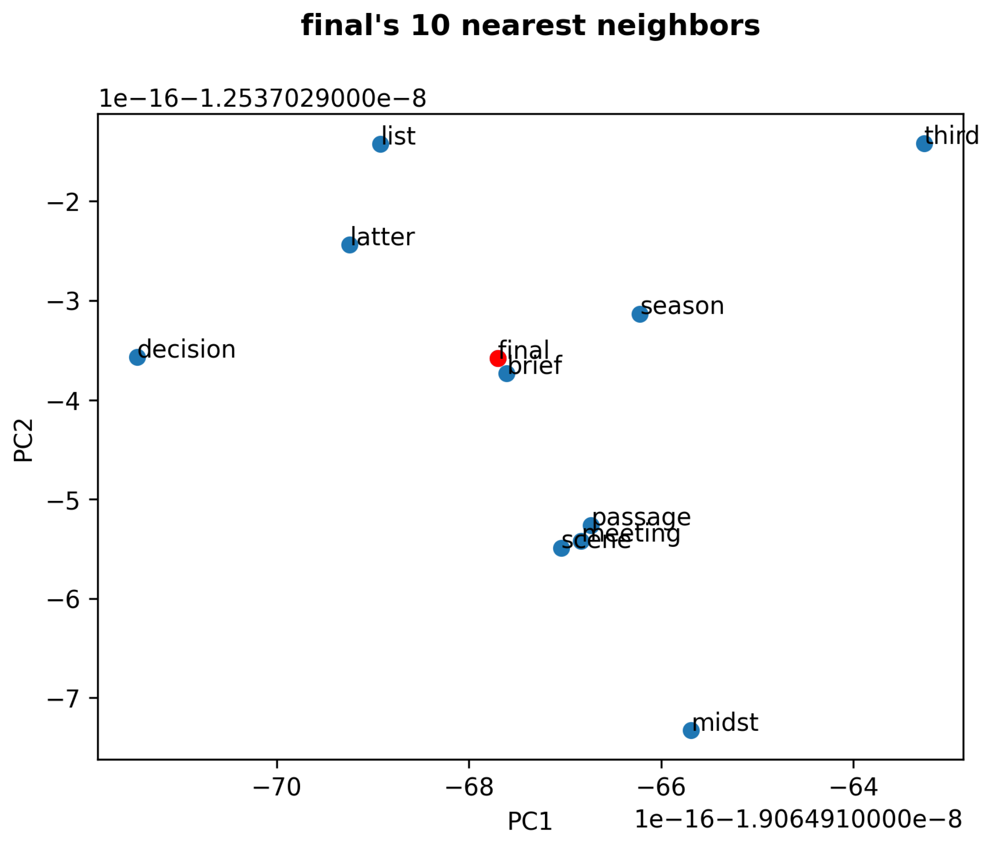

Figure 2: Top 10 nearest neighbors to ”final” in the 2d vector space using PCA

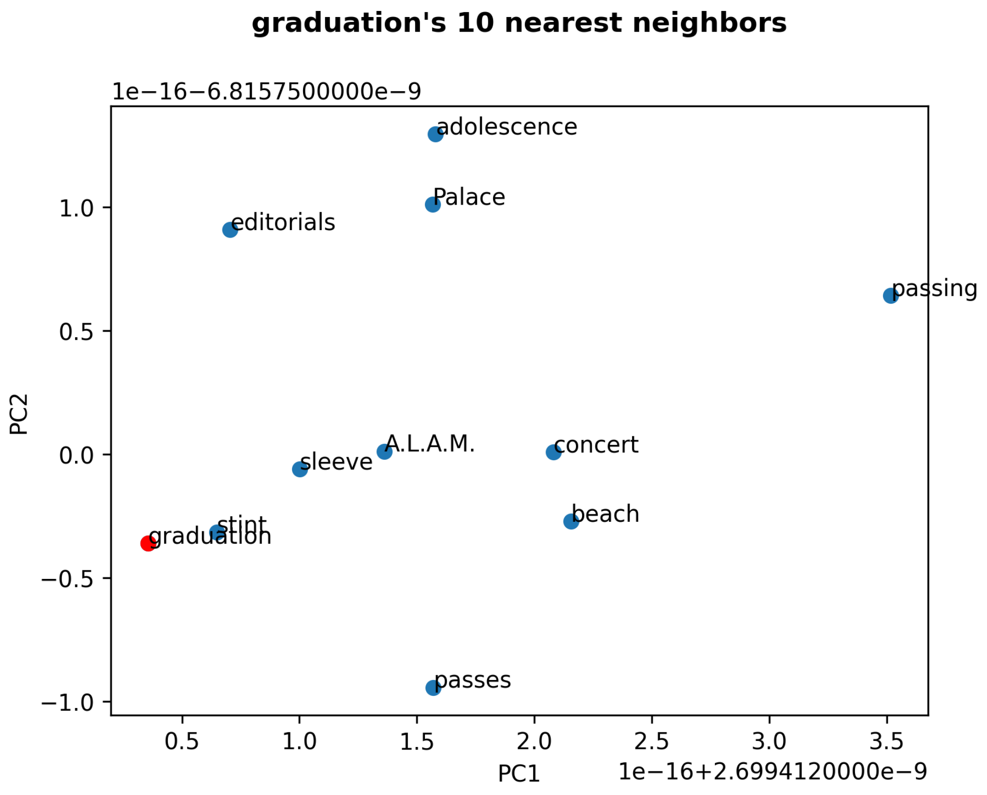

Figure 3: Top 10 nearest neighbors to ”graduation” in the 2d vector space using PCA

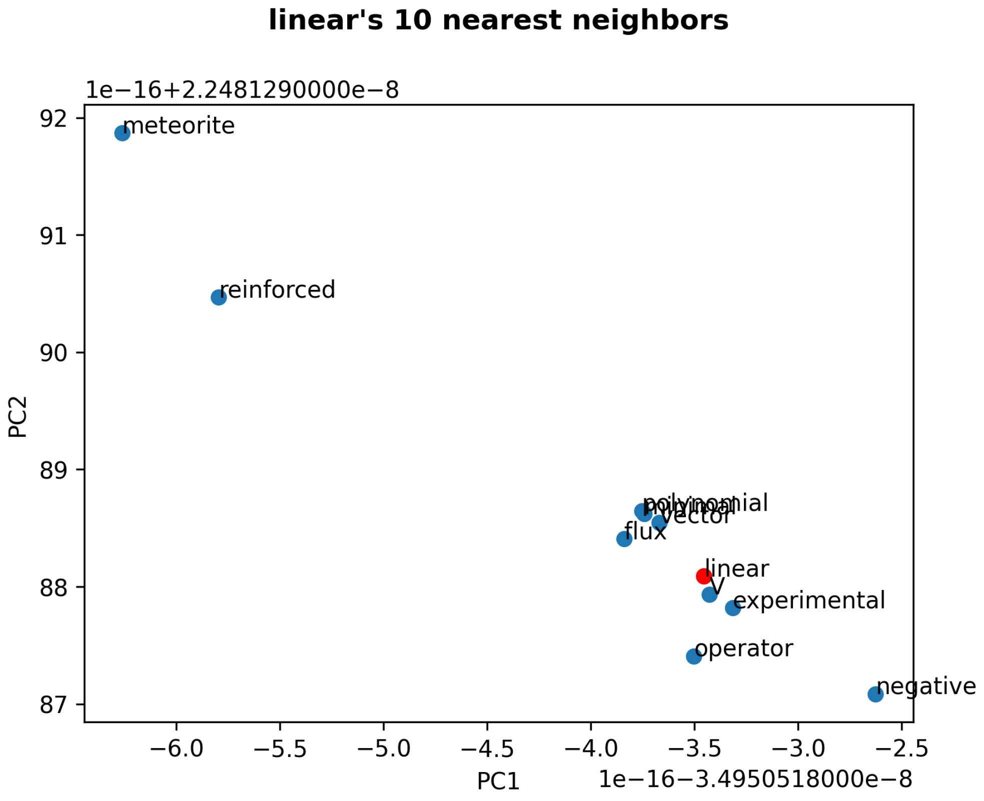

Figure 4: Top 10 nearest neighbors to ”linear” in the 2d vector space using PCA

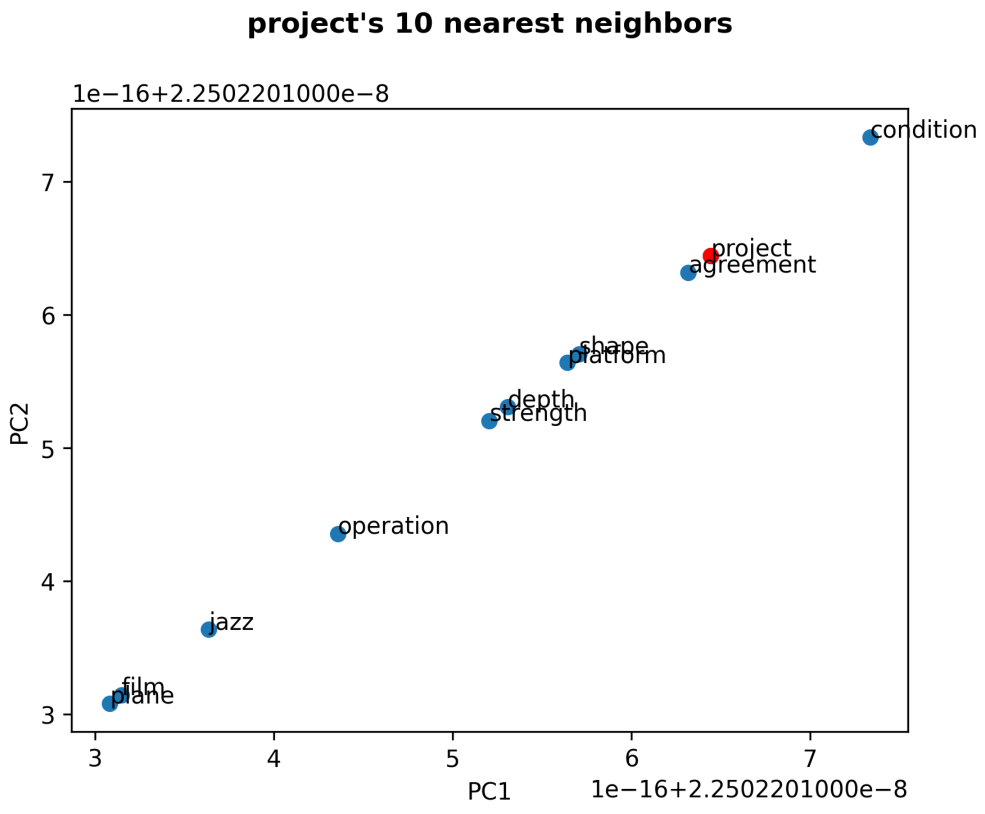

Figure 5: Top 10 nearest neighbors to "project" in the 2d vector space using PCA

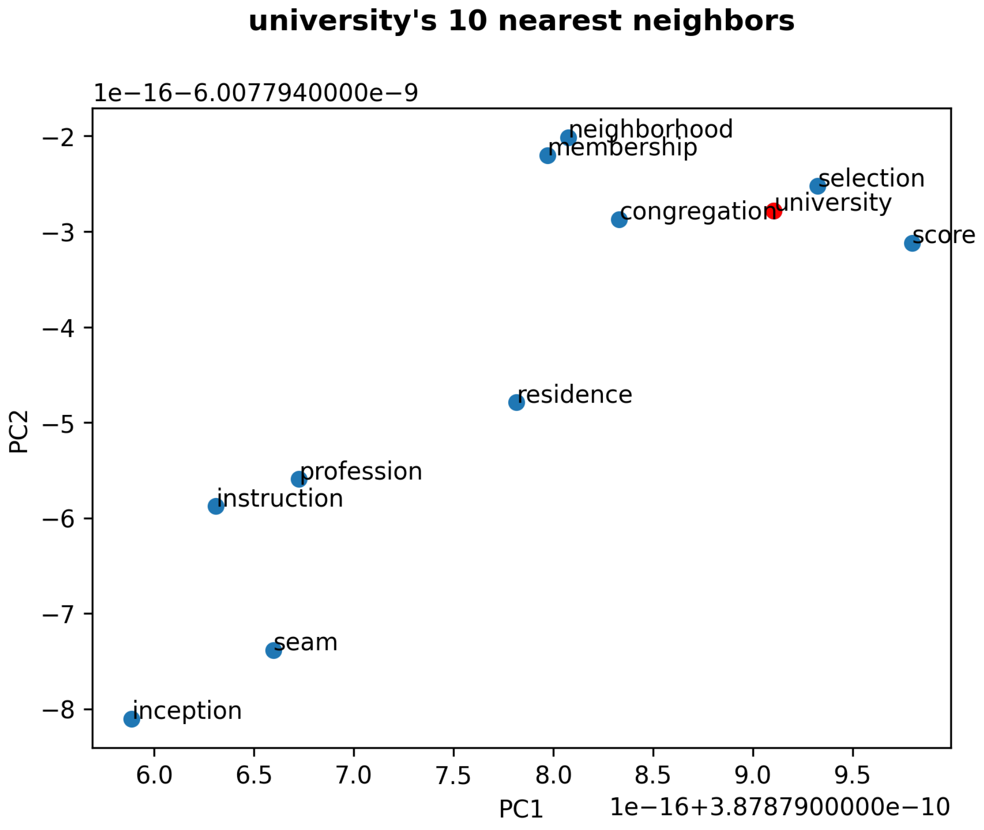

Figure 6: Top 10 nearest neighbors to "university" in the 2d vector space using PCA

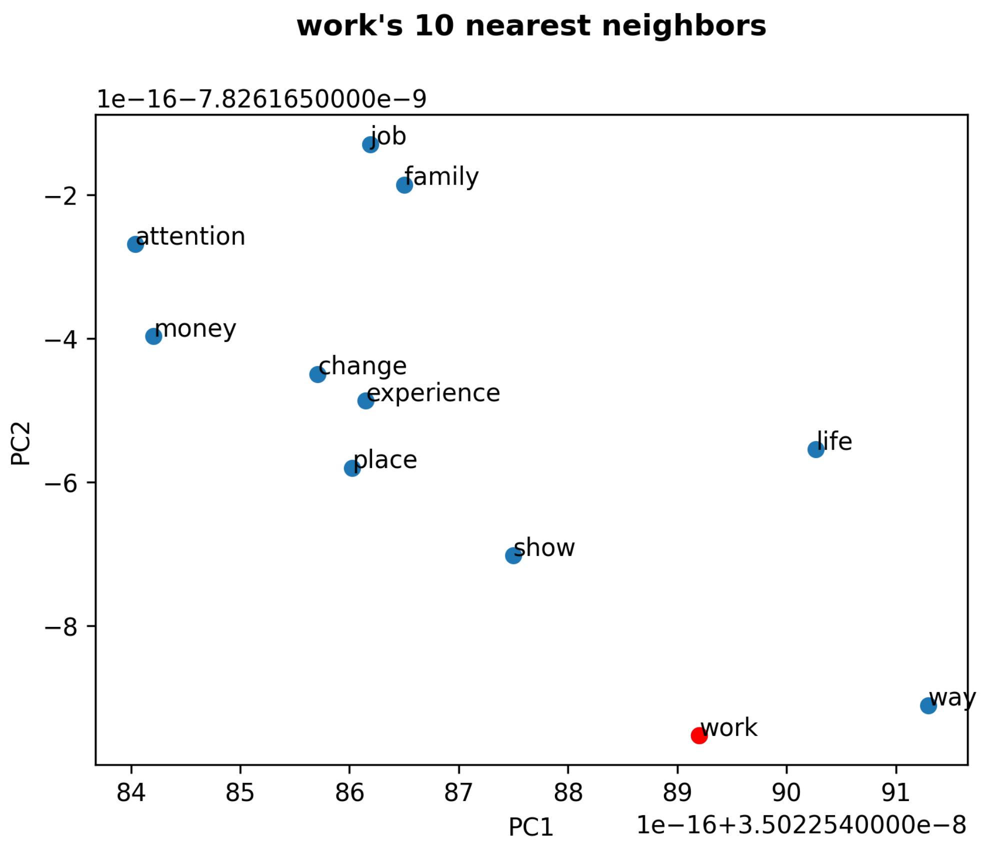

Figure 7: Top 10 nearest neighbors to "work" in the 2d vector space using PCA

References

[1] WORD2VEC CITATION

[2]

[3]