PRINCIPAL COMPONENT ANALYSIS AND FASTMAP ALGORITHM

PART 1 IMPLEMENTATION

PRINCIPAL COMPONENT ANALYSIS

The PCA Algorithm is implemented in pcaparser.py and pca.py. It is called from hw2.py.

First the data is subtracted from their means in order to normalize data. The concatenated matrix is then used to find the covariance, and then that covariance matrix is used to find the eigenvectors.

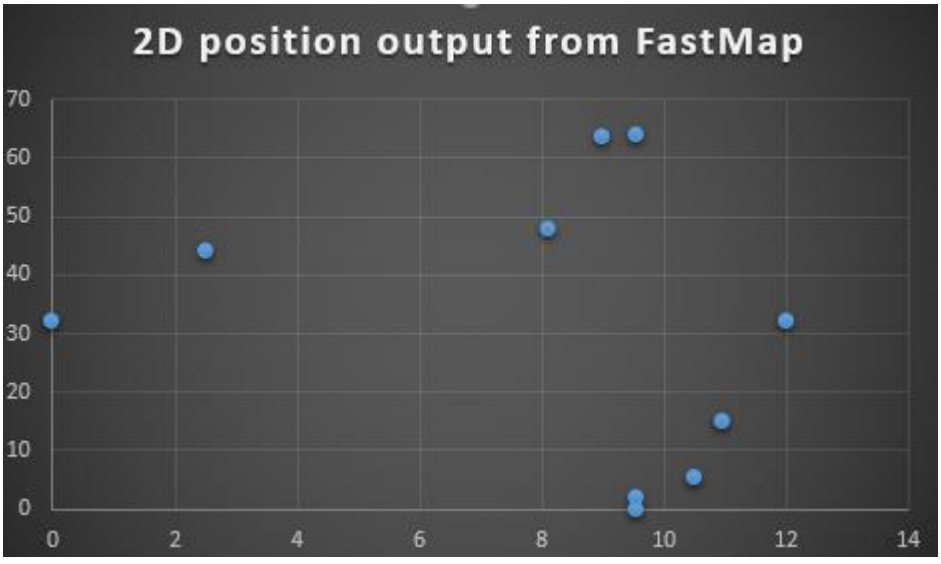
The 3 eigenvectors are sorted based on their respective eigenvalues from largest to smallest. Based on the algorithm the 2 orthogonal vectors that gives the most variance and second most for our data is:

$$\begin{matrix} 0.8667137 \\ -0.23276482 \\ 0.44124968 \end{matrix} \text{ and } \begin{matrix} -0.4962773 \\ -0.4924792 \\ 0.71496368 \end{matrix}$$

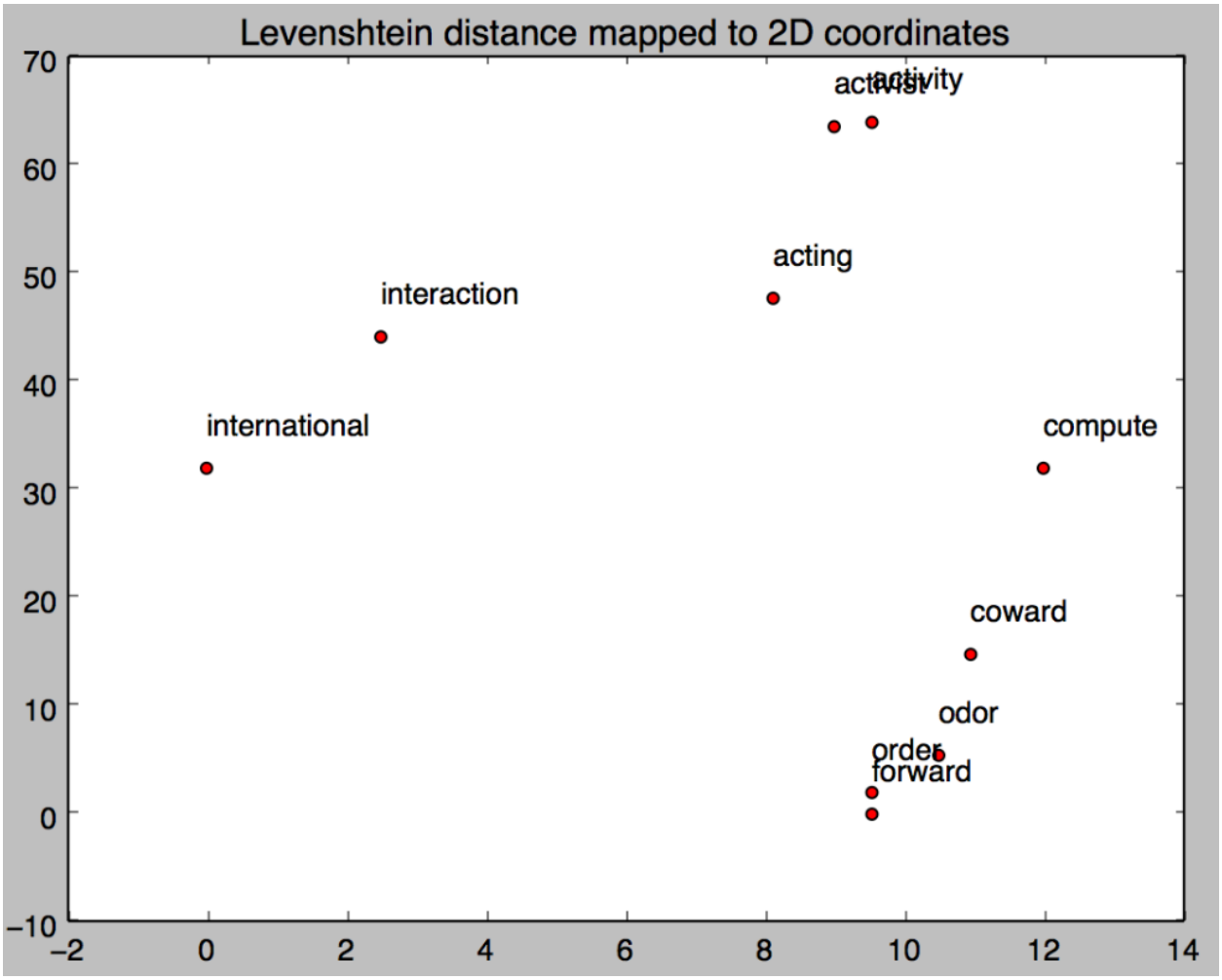
FASTMAP ALGORITHM

The FastMap Algorithm is implemented within fastmapparser.py, fastmap.py, and is called from hw3.py. In order to make the recursive main function for the algorithm cleaner, a FastMap class is created, and many of its transforming data is kept as global variables within the object.

The output of the algorithm, (Images with word index, x-coordinate, and y-coordinate) are written in a csv for further visualization in fastmap-out.csv. I have plotted the data. You can see three roundabout clusters (which can later be found using algorithms such as k-means)



After investigating a blogger's FastMap code (as described in Part 2,) I cannibalized his visualization code, and was able to associate the strings into the plot.

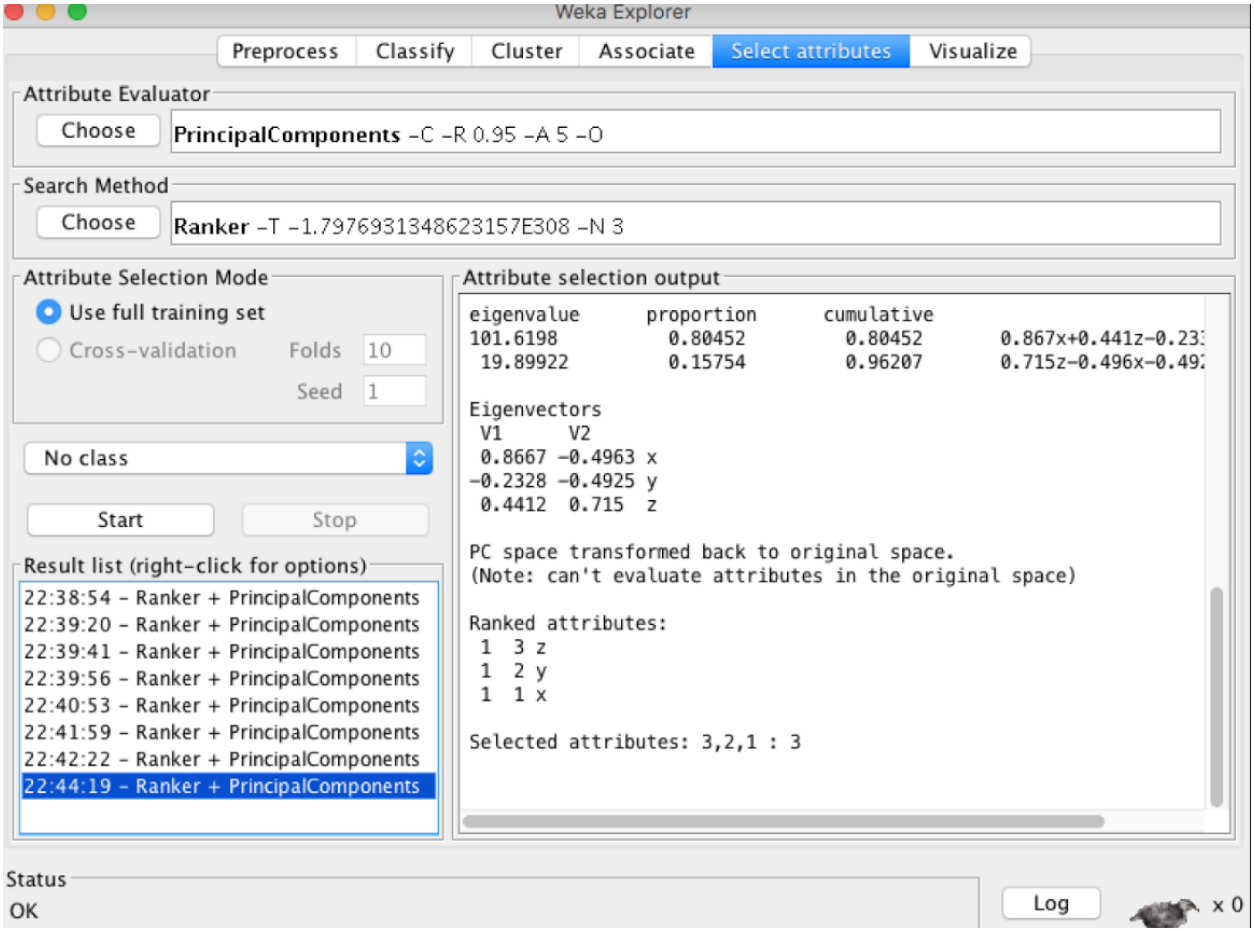


Given the words, I believe the algorithm did a pretty good job at clustering similar words next to each other!

PART 2 SOFTWARE FAMILIARIZATION

PCA

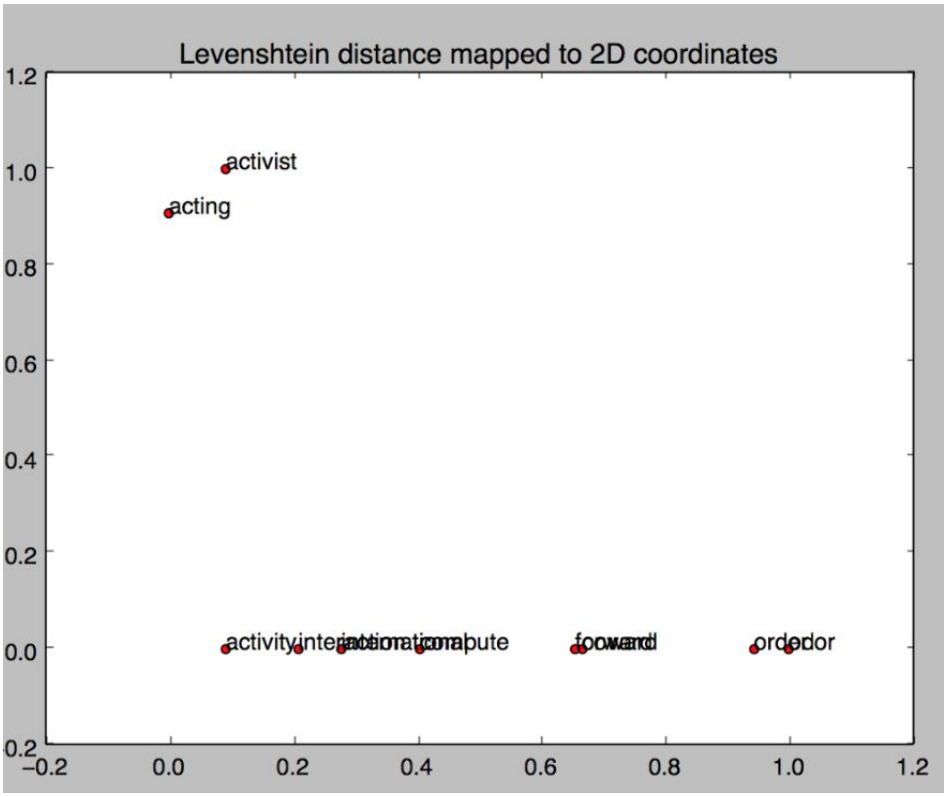
Using Weka, I performed the PCA analysis using the same data as the homework. It is interesting to note that the PrincipalComponents in Weka default settings did not 'center' the data as we have, and the first results did not look like our output. Once the right settings have been set, we get the following results:

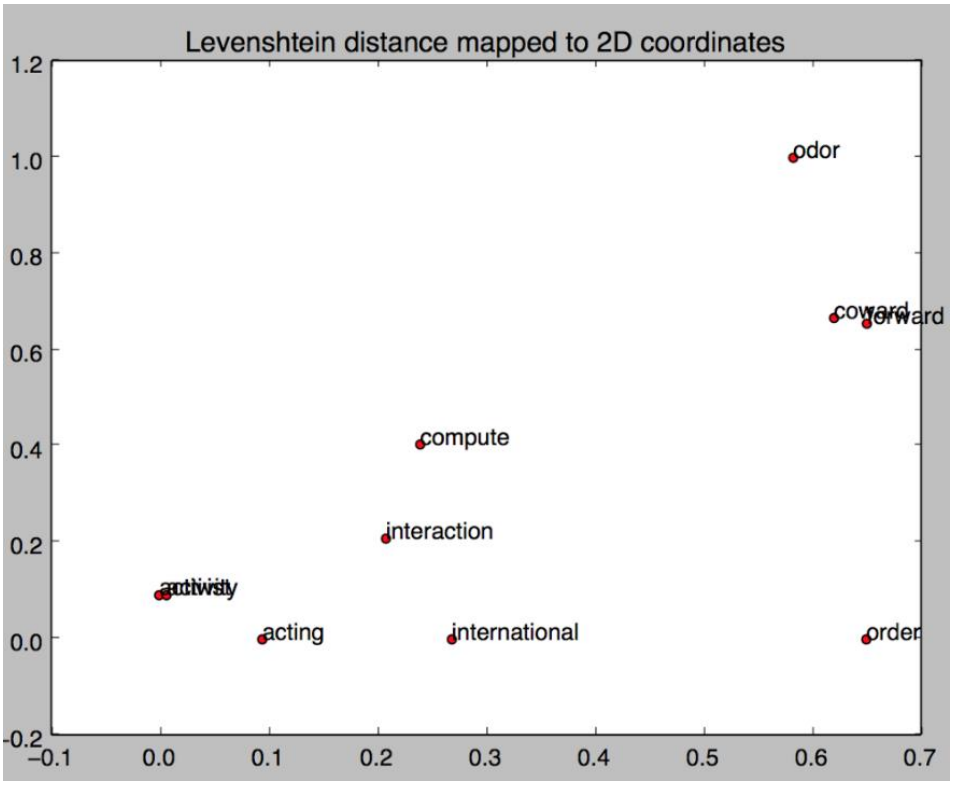


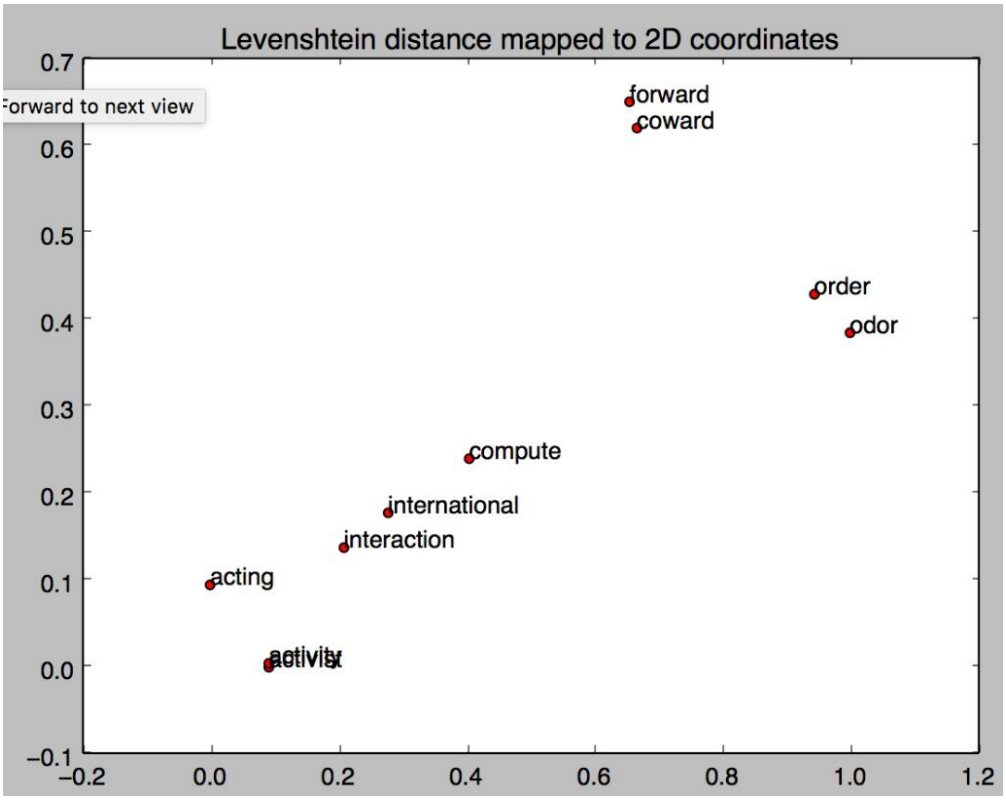
We can see that the result of Weka is consistent with our own algorithm.

FASTMAP

The FastMap algorithm is rather illusive in researching. But, we can take some of the open-source projects found on git-hub, modify is slightly to take our project's data as input, and see what the result is. I found a blogger who had implemented the FastMap algorithm in python as well (http://gromgull.net/blog/2009/08/fastmap-in-python/) Conveniently, he uses the Levenshtein string distance for his experiment as well. I took this blogger's python code, replaced the strings with the 10 words we are given, and tried running the algorithm a couple of times.


Levenshtein distance mapped to 2D coordinates
activist
acting
activity
compute
order
odor
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Levenshtein distance mapped to 2D coordinates
odor
coward
compute
interaction
acting
international
order
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7



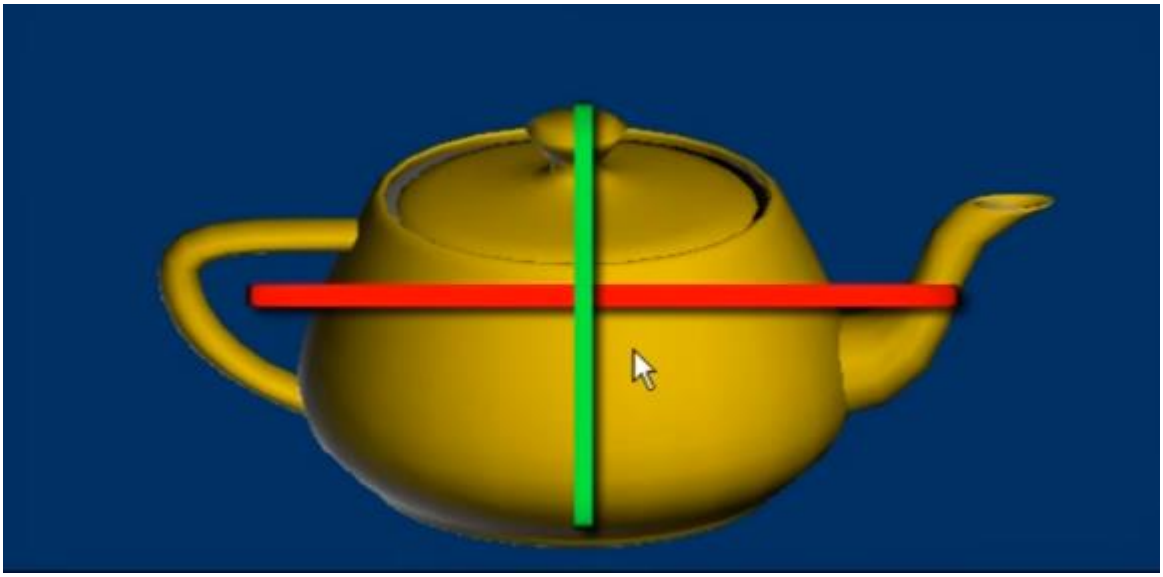
There are 2 major observations to make of his result as compared to my solution.

1. The results from each run seem to be non-deterministic. Sometimes it produces a pretty good result, but sometimes (such as the first figure shown) it produces a result which is quite bad. This is likely because of the way pivots a chosen as they start from a random point.
2. Even when the results return a worthwhile result, the result is quite a bit different from our own solution. However, from visual inspection and human intuition, both solution seems pretty 'good' in terms of similar words being clustered with each other. This is likely because of the fact that the blogger calls upon a python package which calculates the Levenshtein distance ration, and stores the distance matrix, while our own solution parsed a pre-calculated dataset given by the professor. The difference of the actual distance formulae may have caused the results to be slightly different.

PART 3 APPLICATIONS

PCA

One application for the Principal Component Analysis is actually not entirely in the realm of Machine Learning. When given a 3D object (or 3D dataset for that matter) you can use PCA to find the axis in which the points present the most variation. Using these axis, you are able to find the 'view' for those points that will highlight the most amount of features.



FASTMAP

FastMap is an interesting algorithm because it does not hinge on a coordinate system. Thus, you can take rather abstract ideas of distances, and still use FastMap to visualize it in 2D space. Such examples include the distance between words (that is, the amount of edits it takes from one to the other) As mentioned previously, it is also interesting to note that once the FastMap algorithm has assigned the image's coordinates, a secondary layer of analysis such as cluster analysis can be done to produce even better interpretation of data.