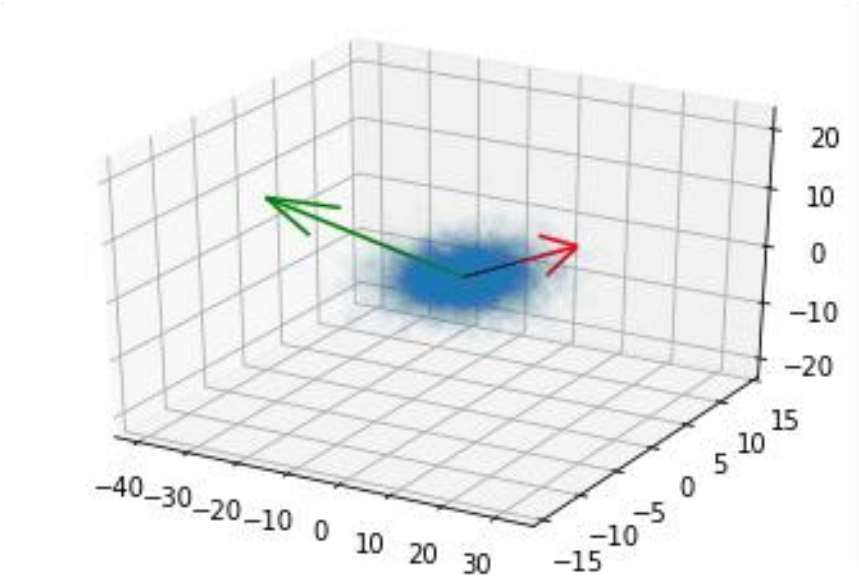
# Assignment 3 Report

(1) PCA: output the directions of the first two principal components.

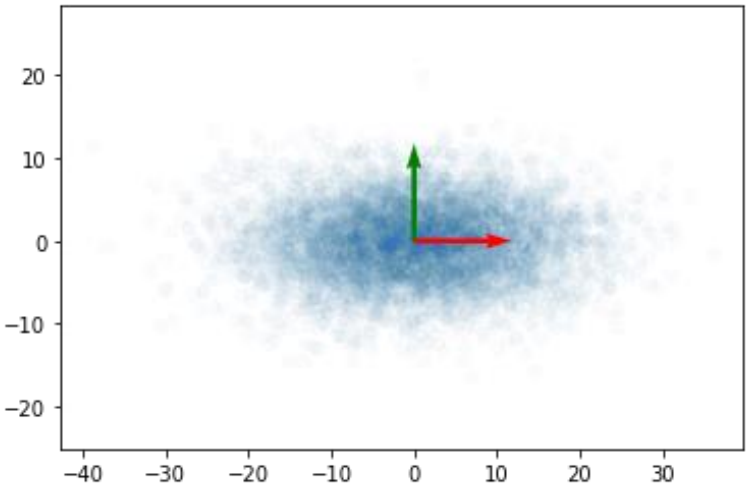
The First Two Principal Directions of the 3D points are the column vectors:

$$\begin{bmatrix} 0.867 \\ -0.233 \\ 0.441 \end{bmatrix} \text{ and } \begin{bmatrix} -0.496 \\ -0.492 \\ 0.715 \end{bmatrix}$$

Visualized in the red and green directions (vectors not to scale for visibility):




Running the PCA and projecting the data onto plane defined by these directions gives us:

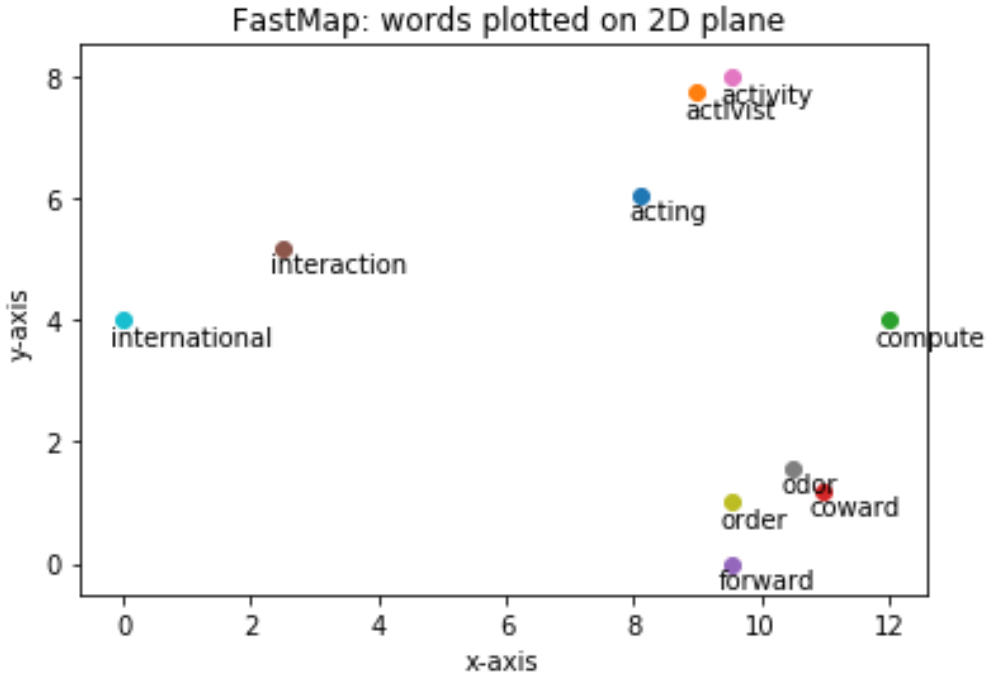


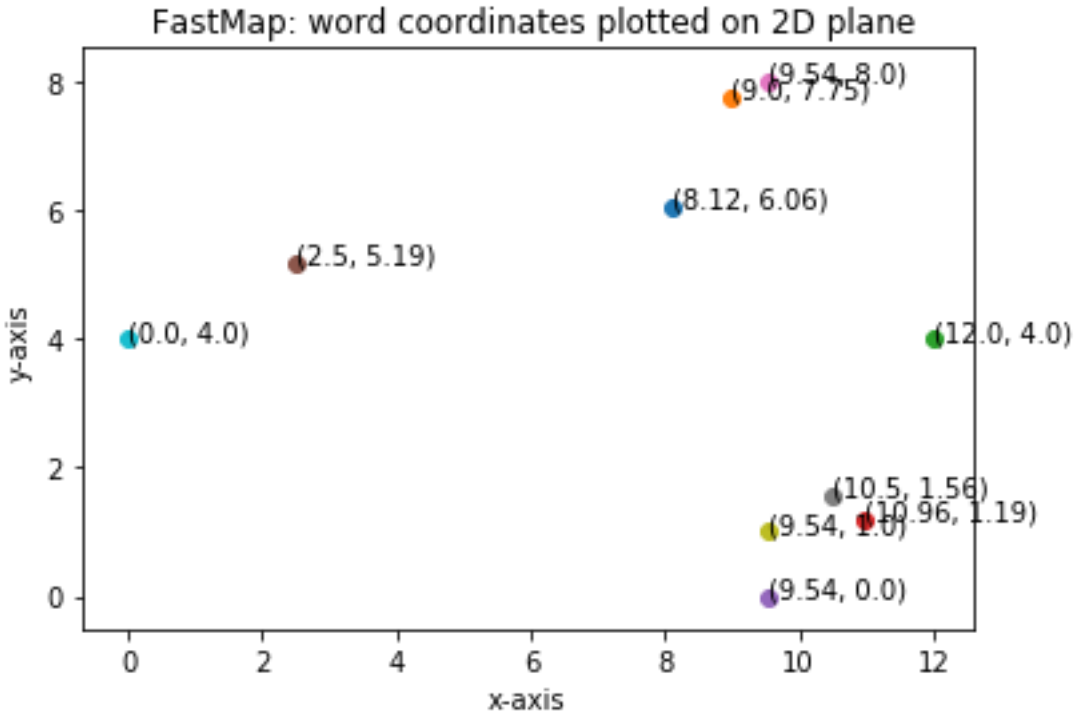

Which has the respective (red & green) 2D principal component directions above: (also not to scale):

$$\begin{bmatrix} 1.00 \\ -7.25x10^{-16} \end{bmatrix} \text{ and } \begin{bmatrix} 7.25x10^{-16} \\ 1.00 \end{bmatrix}$$

(2) FastMap: Plot the words on a 2D plane using your FastMap solution.

Using random.seed(0) and 7 pivots in the step1 farthest pair search, the fastmap implementation gives us the following embedding of the words



*Additional descriptions and challenges:*

The PCA was done using a numpy array and list data structures as well as numpy's linear algebra eigen method. A list of means was stored for each dimension in order to preprocess/demean the data before computing the covariance matrix. The numpy.linalg.eig method was then used to decompose and store eigenvalues and column eigenvectors in an array which was then sorted from greatest to least eigenvalue. This was then used to create another array for the k-largest eigenvectors (U_tr) which was then multiplied by the data to arrive at the projected points stored in the Z array which was then plotted on a plane. For additional visualization purposes the PCA was run a second time on this 2D point cloud to visualize it's principal directions.

The FastMap was also done using mainly numpy arrays and lists. First, a distance function, df , is defined by initializing a symmetric matrix of pairwise distances from the data. One challenge with this was remembering that python is a zero-indexed language so a dummy row and column of zeros was inserted so that the column and row numbers corresponded to the objectIDs. The distance function then called itself recursively depending on which embedding coordinate was being computed and the new distance was computed by subtracting from the old distance (i.e. previous coordinate) the distance amount that was already explained in the dropped axis (i.e. embed) per the formula described in lecture. Next, the getFurthestPair helper function was defined to identify a pair of furthest words in linear time using the heuristic described by the professor and calls the distance function for each coordinate in a recursive fashion so that at each iteration we are finding the furthest pair according to the new definition of distance function. I then run the linear search for the maximum distance using 7 pivot changes since the professor advised that this heuristic method usually converges in around 5-7 iterations. Then the FastMap method is implemented recursively. In the first iteration it finds the furthest pair of objects and then computes the projection line and computes the projected distances of the other objects and embeds them as the first coordinate. Then on the second iteration, the coordinate is incremented and the 2nd coordinate is computed using the new distance which is simply the old distance function minus the previously explained Euclidean distance from the first coordinate. The result is an embedding matrix of 10x2 (# words by k=2 dimensions) where each row represents a coordinate pair that is then plotted in the above FastMap word plot as requested.