Overall strategy

We find vectors that are closest to each other, picking as many as will fit within a specified distance. We pick 4 at first, and check if they are close enough to each other, by the average distance each word is from the others, to share a clue. We set a distance threshold to determine whether to reduce the list by 1 each time. If the 4 words are too far apart, then we try for 3 words, if not, try for 2. Otherwise, we return the clue for the first element in our word list.

For 2 or more words, we compute the vector between them, this is known as our ideal vector. We push the deal vector away from the dangerous words (namely other team’s words and assassin word), to a degree inversely proportional to their distance. Finally, we find the word closest to this ideal vector, ensuring it does not overlap with words present on board, and this is our clue.

Implementation details

Load embeddings: purpose is to create dictionary of all embeddings in file. Able to take in google news 2013 or Wiki English Feb 2017. We were working with the former but runtime took too long because it has a much more massive corpus, so we switched to Wiki, but the function is able to take in google news just in case.

Generate input dictionary - generate a dictionary (where the keys are: our team, their team, neutral and assassin) from a list of actual codenames words sourced online for testing purposes, values are randomly generated. List is shuffled around in order to prevent duplicates.

Spymaster function: This is the main function that works like a human spymaster: takes in the input dictionary, returns the clue, number, and the words it is giving a clue for (for us to check). It splits the input dictionary into respective dictionaries. Gets a list of all board words (to later prevent clue word from being identical to any of the words on the board). Then we populate respective dictionaries: Key is (our, their, neutral, assassin) word, value is the base value. Call setVectors function in order to assign actual vector values to respective dictionaries. Calls on Ideal vector function to find the ideal vector (Ideal vector will later call on other functions that improve upon it)

setVectors: input is condenames words. Creates keys out of words and the values are the respective vectors associated with them by parsing through embeddings (google news/wiki)

getAveDistance : takes in vectors and n as input where n is number of vectors.  Finds average distance between words (helps us determine whether to give clue for 4, 3, 2 or 1 word based on how close they are to each other)

vectorIsCloser: check if possible vector (one of the inputs) has a better average distance than word in our list that is furthest from the rest, used in FindIdealVector.

fndIdealVector : Looks for words that are close together, picking more than two if more fall within a specified distance, and calls calculateIdealVector to find a vector corresponding to the  hypothetical optimal connecting clue.

calculateIdealVector: Outputs ideal vector between input vectors by averaging along each dimension, output is later used in getClue.

Improve vector: return improved version of ideal vector. Logic: V(ideal) = V(target) - V(1/distance btw ideal vector and other team’s vectors) using inverse proportion. We tried using torch to calculate gradient descent but was too difficult, so used this as suggested by Prof.

getClue function: takes in the ideal vector, all word embeddings, all input words and outputs the clue word that best matches the ideal vector. The logic is to create a dictionary where key is the word, value is distance from ideal vector. We find the word with the minimum distance to the ideal vector. Also ensures clue word is not identical to any of the board words.

What worked, what didn’t

Clue is not accurate enough. The words selected by spymaster to give clue for happened to fall under one category –foods, so the more accurate clue would be ‘food’, ‘edible’, ‘eat’, etc. however it gave a clue for a completely different category, hence we tried to tweak the numbers for distance to increase accuracy.

Another time, gave clue “death” for Australia, which does have some connections (Australia known for being home to many dangerous animal and plant species), however “death” happened to be one of the board words so getClue function was edited to account for that.

Gave a clue of “unavem” - turned out to be an acronym for United Nations Angola Verification Mission. Decided to half the size of embeddings so that a lot of redundant words like this acronym can be filtered out.

Found that cosine and Euclidean distance gave about the same results, and we went with cosine as it was tested second.