# Ling 334 Final Project: Codenames Clue Generator using GloVe Embeddings

**Jason Jewell**
Northwestern University
jasonjewell2024@u.northwestern.edu

**Abstract**

For my final project, I created a program to generate clues for the popular board game Codenames. The program uses GloVe embeddings to capture semantic similarity between words, and find clues that are optimal given the words on the board and their colors. The program performs moderately well, with most of its clues imitating how a human would play the game. However, the word embeddings it uses do not always align with human intuitions of similarity.

## 1 Introduction

In Codenames (Chvátil), there are two teams, red and blue. On each team there is a 'spymaster' and any number of 'operatives'. There is a board consisting of 25 words or concepts, some of which are red or blue. Only the spymasters know the colors of the tiles, and must take turns giving one-word clues to guide their operatives to pick out the tiles of the correct color, while avoiding those of the opposite color and those which are not colored. Whichever team's operatives successfully pick out all of their color's tiles wins- unless a team picks the one 'assassin' tile, which immediately causes them to lose. The strategy of the spymaster must be to figure out a clue that relates as many tiles of the correct color as possible in each turn, while avoiding similarity to any of the opposite color or the assassin. The semantic nature of this task makes it a good candidate for being simulated with word embeddings, which can successfully capture very useful notions of semantic similarity. I used GloVe vectors (Pennington et al., 2014) to create a program that, given a Codenames board, attempts to generate the best clue for a team's spymaster.

## 2 Data

I utilized the following outside data in my program:

**Pre-Trained GloVe embeddings:** I used GloVe's pre-trained Common Crawl vectors, which were trained on 42 billion tokens and each contained 300 dimensions. (Pennington et al., 2014)

**Keith Vertanen's *Big English Word Lists*:** Vertanen has compiled large lists of English words on his website, by comparing other corpora and including words that appeared in at least *n* of them (Vertanen, 2018). I chose his list of words appearing in at least 8 lists, which contains 108947 words. I used this list to filter the GloVe vectors and only use the vectors for these ˜100k most common words. There were previously ˜1.9m vectors, the vast majority of which were not useful for the task, as they would never be used in Codenames, or are not even words.

**Function word list:** I used a list of function words from James O'Shea's blog *Semantic Similarity* (O'Shea, 2013). I used this to filter out English functor words from being given as clues, since they often have high vector similarity to words without carrying much semantic information that could enable a player to correctly guess. I slightly edited the list; namely I removed the words *first*, *second*, and *third*, and added the word *better*.

## 3 Methods

To use the GloVe embeddings in my code, I transferred over my Assignment 5 (Semantic Similarity) code, specifically the embeddings.py module. This module contains code to load in word embeddings and calculate cosine similarity.

To generate random Codenames boards, I adapted a program by HBiede that accomplishes this (HBiede, 2020). The program was originally written in Ruby; ChatGPT translated it to Python for me (OpenAI, 2021). I also included the option for the user to alternatively input their own board, if, for example, they wanted to use the program during an actual physical game (something which I did several times during testing).

To generate a clue, the program assigns a score to every possible clue from the ˜100k words it considers, and returns the word with the highest score. The score is meant to roughly capture "how many tiles of the same color are closer than any of the opposite color?", while also adjusting for details that proved important during testing. The score is calculated as follows- first, the remaining words on the board are all ranked by the cosine similarity of their vectors with the vector of the clue. The program then looks at this sorted list, from the top most similar, one word at a time. For each word, the score is adjusted based on the color of the tile as well as the similarity to the clue. If it is the same color as the guessing team, and it is sufficiently similar, the score is increased by 1, as well as a small amount proportional to the similarity (this is to favor more similar words, if all else is the same). If it is the same color but it is *not* sufficiently similar, the score is only increased by that small amount (in this case, we don't want a large increase in score, but we still want to favor a similar clue where a top word is slightly more similar). If the word has no color, the score is decreased by 0.5, to penalize neutral words being guessed. If the word is the assassin, the score is decreased by 1 and the algorithm stops checking more words. If the word is the opposite color, the score is decreased by a small amount proportional to similarity, in order to favor clues that are farther away from the most similar opposite color word; and the algorithm also stops checking more words.

Finally, once all possible clues are scored and the top clue is selected, it also must give a number representing how many tiles it is trying to communicate, as a spymaster is supposed to. The program achieves this by simply counting how many tiles of the correct color are more similar than any tile of the opposite color (or assassin).

After inputting or randomly generating a board, the user can generate clues for either team, remove any words from the board, or check the similarity of a clue to all words on the board. The last option is useful to see 'what the bot was thinking' for any clue it gave.

## 4 Results

The program is definitely good, some of the time, at generating human-like clues for a given board. For very small boards, it is easiest to see this. For example, when tasked with relating the words *cow* and *leg*, it successfully finds the word *calf*. When tasked with relating the words *red* and *blue*, while avoiding the word *green*, it gives the clue *purple*. For larger boards, it usually tries to connect one to four words. Five words or more is usually impossible even for a skilled human player without any special luck, so this makes sense. I played several rounds of Codenames with my friend using this program to play against myself- one person played as one spymaster, and the program played as the other, while the other attempted to guess for both teams. The program is definitely not as skilled as me or my friend when it comes to generating guesses, but it's not bad either. It couldn't win a game, but it did come close.

There are definitely some issues and quirks. Some of these can be attributed to my program, but some of them are undoubtedly just due to inconsistencies between the word embeddings and human intuitions of similarity. For example, the word *hand* is, for some reason, very similar to a lot of words according to GloVe. Whenever the word *hand* appears on a board, almost every clue the program gives is 'trying' to point it out, even when I can't see a logical connection. Other words also have very high similarities with words that seem mostly unrelated- *mercury*, for example, is most similar to *mariner* and *parts*. The program once gave the clue *parts* for the tile *mercury*, and while this made no sense to me as a human Codenames player, the program had no reason to think I didn't see these words as incredibly related.

## 5 Discussion

This method of generating Codenames clues, using GloVe embeddings, can create a decently skilled bot that often creates a guess that a human would reasonably give. There is definitely a limitation to how good this program could be, just given the word embeddings that I used and their often illogical 'similar' words. In a future project, it would be interesting to see if other methods of capturing word similarity, such as a knowledge base of semantic relations or category information, would perform better at generating clues that can be easily interpreted. I suspect that they would be- in

Codenames, a go-to strategy is definitely to find a specific category that several words of your color belong to. The word embedding approach is decent, but fails to really maximize the amount of information that can be conveyed by a skilled cluegiver, who must find complex connections between words- connections that go deeper than just appearing in similar contexts.

## References

Vlaada Chvátil. Codenames.

HBiede. 2020. Codenames-Board-Generator. Original-date: 2020-05-06T01:09:09Z.

OpenAI. 2021. ChatGPT. `https://openai.com`. Accessed: June 3, 2023.

James O'Shea. 2013. Function word lists.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Keith Vertanen. 2018. Big English Word Lists.