

# CS 571 Final Project Part 3:

## Codenames AI

Oleg Aprelikov, Shaurya Mathur, Aayushi Neema, Rohan Purandare

### Formal Model

#### Minimax Approach

After creating the initial graph that contains the relationships between each word, we modeled this problem using a minimax algorithm. As this is a turn-based game, each move for a particular user will impact the next turn for the other user. As such, each node in our minimax tree represented different board states depending on how the game could be played. Using the weights from the word association graph, we determined which boards are more favorable than others. Coupled with this, we also used the weights from the word association graph to determine how likely the field operative is to guess certain board permutations. As this is a 5 by 5 board, there are several board permutations.

#### Randomized Approach

The goal of this approach was to simulate hundreds of games for each potential clue that the bot can give as a spymaster, and to pick the clue with the highest win rate. This is a similar approach to [Monte Carlo tree search](#). In Codenames, there shouldn't be any subtle lines of play that completely change the result of the game, so randomized simulations should yield respectable results.

To begin the simulation, a weight  $w_i$  was assigned to each clue  $i$  based on how good the clue appears to be locally. This is discussed in more detail in the tasks section. After assigning the weights, they were transformed into probabilities using  $P(i) = \frac{w_i}{\sum w_i}$  where

$P(i)$  is the probability of picking clue  $i$ . Utilizing these probabilities, a random clue was picked as the spymaster's turn. A similar randomized procedure was performed to simulate the field operatives' turn. This process of randomly deciding what the spymasters and the field operatives do was repeated until the end of game at which point the result of the game was recorded.

Finally, to decide which clue the spymaster picked, every clue was iterated over and the simulation above would run  $k$  times after picking that clue. The clue with the most wins was picked. To speed up this process, particularly bad clues were skipped during this process

## **Word Association Vectors w/ TF-IDF Vectors and Cosine Similarity Matrix**

We used online documents like Wikipedia and the concept of TF-IDF to create semantic word associations, which served as the search space for our formal model. For our word associations:

1. For each of the 400 possible words that can appear in a Codename board, find the Wikipedia article (using Wikipedia Python API) whose topic is that word, a set of definitions for that word (via a dictionary), and a set of synonyms for that word (via a thesaurus)
  - a. For some words like Amazon, the wikipedia page summary links to various other wiki articles such as “Amazon River”, “Amazon Rainforest”, “Amazon (Company)” etc. In this case we go one extra level and parse the additional hyperlinks. That way, we make sure all words are comprehended in all contexts in our vocab associations.
2. **Use web-scraping and data-processing techniques like tokenization and stopwords removal** for every Wikipedia article create a large set of vocabulary words
3. For each vocabulary word, **calculate the TF-IDF vector**
  - a. Each board word is represented as a TF-IDF vector
  - b. The value in dimension  $x$  is the TF-IDF score of that vocabulary word in document  $X$ , therefore it's association to the board word  $X$
4. For each pair of boardwords, we use their TF-IDF vectors to **calculate their cosine similarity, and store the result in a similarity matrix** (where there's a row and column for each word)
5. Therefore we use these 2 main data structures:
  - a. Cosine similarity matrix to group board words into similar categories and then find a guess which collectively could relate to one of these.
  - b. TF-IDF vector to check how relevant the vocab word is to a guess word to be able to provide the vocab word as a clue.

To generate the clue given by the Spymaster bot, and to determine the guess made by the Field Operative bot, we used the 2 formal models described in the section above.

## **Tasks**

## **Key tasks for creating the word associations**

- Obtain a set of source words that will be the knowledge base/source for all playing words on the board to be guessed. For this we used an existing repository of English words. That list can be found [here](#).
- Find a document for all relevant vocab words for our associated word bank. For this we utilized the python wrapper on [Wikipedia API](#).
  - We source all Wikipedia pages with the article title matching the word from the word bank. We then use the summary portion of the article to build the TF-IDF vectors. If the summary section is not adequately long enough, or includes links to other Wiki articles that the word could reference to, we use the hyperlinks property of the API response to scrape the linked articles as well.
- After having the entire corpus of wiki articles for all words, we create the TF-IDF vectors for each word's documents.
  - TF-IDF is an already popular industry algorithm which stands for "Term Frequency - Inverse Document Frequency" which can reflect the importance and relevance of a word in a document. In our scenario, we associate the importance of a vocab word to a particular board word as the TF-IDF score of the vocab word in its Wikipedia page among all words.
  - There are multiple online existing libraries that can create TF-IDF scores directly, but we build our module from scratch, by parsing the corpus and calculating the frequencies.
  - There are 2 parts to the TF-IDF score:
  - TF (Term Frequency): The Term Frequency measures the frequency of the word in the document and is calculated by dividing the number of occurrences by the total number of words in the document.
  - IDF (Inverse Document Frequency): Inverse document frequency measures how common or rare a word is across the entire corpus, thus in our case across all wikipedia articles. It is calculated using the ratio of the total number of documents in the corpus divided by the number of documents that contain the word. Since we work with a large corpus, we take a log of that and thus add 1 to the denominator.
  - We then multiply the TF and IDF scores to get the TF-IDF score.
- The second key structure is the cosine similarity matrix. We do so by using a python library in the NLTK library, which takes their TF-IDF vectors from the earlier bullet point. This allows the algorithm side of choosing words to group similar words together and look for clues that can be used for multiple words.

## **Key tasks for creating the Minimax Algorithm**

- Quantifying the quality of clues: Once we had created the structure of the Minimax tree, we then had to decide how we would account for the fact that the AI is playing with a teammate. We needed to determine the likelihood of our teammate guessing the words that our AI gives as a clue and we decided to use the similarity values from the word association vectors to quantify that. Additionally, we had to give different weights to the different kinds of words that appeared on the board to represent the benefit of choosing certain words and multiplied them by the similarity between the different words to ultimately generate the benefit of the clue. We decided that words that matched the AI's team color would receive a coefficient of 3, words that matched the opposing team would have a coefficient of -3, words that were neutral would have a coefficient of -1, and finally words that were game-ending would receive a coefficient of -100. Using this, we were able to quantify the quality of the clue and how much it would be able to help our teammate guess the word(s).
- Factoring in our teammate's guesses: As the words on the board are assigned randomly, it is likely that the two teams have similar words. As such, it is possible that a clue given can be valid to words for either team. Per the rules of the game, when a player guesses a word that does not belong to their team, the turn ends. To incorporate this into what the user would guess, for each clue, we sort the words that best match the clue given and expect that the AI's teammate guessed in that order, terminating when they guess a word that does not belong to their team. For example, if the clue's best matches in order were Red<sub>1</sub>, Red<sub>2</sub>, Red<sub>3</sub>, Blue<sub>1</sub>, Beige<sub>1</sub>, we would imagine that our teammate would guess Red<sub>1</sub>, Red<sub>2</sub>, and Red<sub>3</sub>.

## **Randomized Algorithm**

- Assigning clue weights: The first thing that needs to be done is to estimate how good each clue is by assigning it a weight. This is a similar task that the minimax approach had to solve, but we used a different solution for the randomized approach. A weight of 0 is assigned to each clue that has a higher association with a word of the wrong color than a word of the correct color. This essentially filters out useless clues. The other clues receive a weight based on how much room for error there is and the number of words the clue guesses.
- Simulating field operatives: In order to simulate lots of randomized games, we need to model how a field operative might respond to any particular clue. This is impossible to do perfectly, but we estimate how a field operative might pick a word by assigning a weight to each word on the board based on how well it

matches with the clue. These weights are calculated by taking the association between a word on the board and the clue and taking it to some constant power. This makes large associations more likely to be picked and small associations less likely to be picked.

- Implementing game rules: Finally, all of the game rules had to be implemented in order to actually simulate randomized games. In particular, Codenames has a lot of rules regarding which clues can be used and which ones can't. Unfortunately, it's hard to implement these rules as they often rely on subtleties of language. Therefore, our implementation will sometimes give out a clue which would technically not be allowed. However, this is rare in practice.

## **Outcomes**

### **Word Association Outcome Analysis**

There are multiple industry standard word associations such as GloVe, WordNet, and Word2Vec that we can compare our structure to for performance measures. GloVe puts emphasis on the importance of word-word co-occurrence across the entire corpus (knowledge base). In GloVe, the word matrix is represented by each row denoting the target word and the columns represent the context words where the context word is the word that the target word appears with, and the values of the matrix is the number of times the word co-occur. WordNet is a lexical, semantic network which does not use vectors however organizes synsets (set of synonyms or words sharing common meaning) in a hierarchical structure. Therefore it includes all word associations and word subset connections like connecting "rice" and "crop" since rice is a kind of crop. WordNet also includes different sections for nouns, verbs and other figures of speech and creates relationships with objects between them. Word2Vec is a popular embedding technique which somewhat similar to GloVe creates dense vector representations of words to create co-occurrence relationships. Most Word2Vec implementations are used to predict target words from surrounding words therefore the use case is slightly different. Unlike GloVe which is focused on matrix factorization, Word2Vec is backed by training shallow neural networks.

However, our word association performs inferior to these state of the art methods. These methods have been developed over multiple years with high amounts of training data and development. Our word association succeeds at correctly breaking down documents into its key words and connecting various documents of different context to the same word (such as Agent (a travel agent), and Agents a finnish band), however it is unable to assign proportionate scores to clue words which statistically differentiate

them from more common words, thus the AI algorithms are unable to choose the best word to connect multiple words.

Additionally, our self-built word association structure has limitations, the first being compute power for creating the TF-IDF vectors and parsing wikipedia pages. Our word-bank of 400 words requires parsing of up to 600 different wikipedia articles, each averaging 4000 words following which we process the texts for stopword removal and performing TF-IDF calculations for the entire corpus and creating 150,000 potential guess words. As such, it takes up to 30 minutes of run time to instantiate this due to our low computing power on personal laptops which limited our testing during development as we worked with a smaller set of articles. Similarly, wikipedia articles are often written with a certain style formatting which reuses similar words across documents, therefore our word association which relies simply from these articles is not fully able to emphasize the uniqueness of words.

### **Minimax Outcome Analysis**

Codenames is a pretty unique game in that the interactions between the two opposing teams is actually rather minimal. Unlike games like Tic-Tac-Toe or Connect-4 where every single move is guaranteed to impact how the opponent plays in the following turn, that isn't necessarily a given for Codenames. While there is still some user interaction, the difficulty of the game comes with communicating with your teammate(s). As such, there was not that much difference in the different branches of the Minimax tree. This led to unnecessary computations which slowed down the overall process.

Additionally, due to the randomness of the words on the board, generating clues was rather difficult as we had a tradeoff between runtime and benefit. Our word association vectors were able to give us a set of words for each board word that had a similarity above a certain value. While this is helpful, when we came across more niche words, it became very difficult to find appropriate clues to give as we did not have that many words to consider. An alternative to this would have been to select the top x words for each board word. That would have guaranteed that each board word would have x potential clues. The drawback of this however is that it would have been an incredibly expensive action. In our word association vectors, we have approximately 150,000 potential clues. So, if for each board word at each level of our minimax tree, we had to iterate through 150,000, it would have significantly slowed down the program.

### **Randomized Algorithm Outcome Analysis**

The biggest limitation for the randomized approach was the lack of good clues due to how TF-IDF works. Almost all of the good clues target only a single word, which greatly limits the usefulness of this approach. The reason why only single words are being targeted is simply because targeting more words is hard and TF-IDF is not good enough for this. Although this approach has some potential, it is extremely hard to evaluate it given the circumstances.

In rare cases when there is a clue that targets more than one word, the randomized AI tends to pick it even if it's not very good. It seems to overestimate how good the field operatives are at guessing. The effects of this problem could be reduced by fine-tuning how the field operatives are simulated, although actually doing so resulted in the problem going the opposite direction.

### **Analyzing the algorithms against each other, and other algorithms in general**

When we were initially deciding how to approach creating a Codename's AI, we had thought of a couple of different ways to solve this problem and had ultimately narrowed it down to using a minimax algorithm or a randomized algorithm to generate clues. At the time, we were not sure which AI would be more effective, and as such, we decided to create them both to test them against each other. Upon comparing them to one another, we realized that the randomized approach took considerably less time than the minimax approach. Both algorithms seemed to frequently provide one word clues (clues that asked the teammate to only guess one word) and both also struggled to give clues for niche words. As such, from a quality perspective, it seems as though the two algorithms were somewhat competitive with one another. However, in terms of efficiency, the randomized algorithm approach was better. As both of the AIs were running on the same word association vectors, we inevitably ran into some invalid clues – clues that broke the rules of the game.

From our initial research on this problem, we saw that this problem had been solved a couple times before in different ways. One technique, created by David Van Dyke, used a forward search algorithm to process different potential board permutation. In addition to this, Van Dyke only allowed his AI to view its teams words with the logic the enemy words will be guessed and consequently removed during the guessing process. While it is difficult to compare the two models since they use two different word association resources, it seems as though our models have considerably less risk than that of Van Dyke. As both the minimax algorithm and the randomized algorithm factor in every single word on the board, our AIs ensure that no enemy word will be clicked.

Another implementation created by David Kirkby involved thoroughly training an AI model to then provide clues based on word clusters created by DBSCAN. With this technique, the machine learning model was trained for 30 hours to learn the relationship between different words. As such, it was able to have a much more sophisticated set of vectors that were far more accurate. Because of a higher guarantee between the relationship of the words, this AI was consequently able to simply group its words in an optimal way to minimize the total number of clues given in general.

## **Improvements**

- Better structure for word associations. The current dimensions of our word similarity model creates vector and word scores for each possible board word with all possible vocab words. In hindsight, it would be a better structure to model the word associations differently where we can connect and get a score for multiple board words and their connection to the same vocab word. Currently, board words were grouped on cosine similarity which was determined from the entire corpus due to which we can't determine the similarity between board words on single potential clues.
- Detailed scoring function. TF-IDF scores are not normalized which causes low TF-IDF scores for most words when working with a large corpus. This way it is hard to set a benchmark metric in the clue decision making algorithm which causes the approach to change from selecting clues that, for example, have a higher than 80% match to selecting top five scores, even if those top 5 words may sometimes be irrelevant words to the board words.
- Utilize better web scraping techniques. Current algorithm removes stop words, punctuations and lemmatize words but still adds plenty of common words which do not enhance the quality of the word vector in the tf-idf setup, thus lowering the metric scoring of other impactful important words which occur just as frequently in the article. (Example: play and use occur just as frequently in the wikipedia article for "Football" but clearly play adds more value to the vector mapping than use)
- In terms of the minimax algorithm, one improvement that could have been made would have been to figure out a better and more consistent way to represent how the AI's teammate would play. We are currently assuming that the AI's teammate will be choosing the words that are closest to the clue in that order, and that is the method that we use consistently throughout the entire algorithm. We could have adjusted the algorithm as the player played alongside the AI to work to create clues that were better focused towards the AI's teammate.
- From our initial research, there were many techniques that we saw and while all of them were pretty different from what we implemented, we could have



incorporated some of their strategies into our algorithms. For example, Van Dyke used a forward searching strategy which could have been coupled with the minimax algorithm.

## Code








**Project Code Repository:** [shauryamat20/571\\_Final\\_Project](#)

We've added the professor (bbullins) and TA Site Bai (best99317) as collaborators on repositories for code access.

### Manage access

Add people

☐ Select all Type ▾

<input type="checkbox"/>	 <b>aayuneema</b> Collaborator		Remove
<input type="checkbox"/>	 <b>bbullins</b> Awaiting bbullins's response	Pending Invite 	Remove
<input type="checkbox"/>	 <b>best99317</b> Awaiting best99317's response	Pending Invite 	Remove
<input type="checkbox"/>	 <b>Oleg Aprelikov</b> Robochu • Collaborator		Remove
<input type="checkbox"/>	 <b>Rohan Purandare</b> rohanpurandare • Collaborator		Remove