Algorithms for Codenames

Dan Friedman and Abhishek Panigrahi
Department of Computer Science, Princeton University
{dfriedman,ap34}@cs.princeton.edu

Fall 2021

Abstract

Word reference games provide a setting for exploring the computational models that underlie human similarity judgments. In this paper, we explore algorithms for the popular reference game Codenames. We describe the optimal solution to the computational problem, and then develop a range of algorithms for approximately solving the problem under different resource constraints, including (a) constraints on the data structures and primitives for calculating word similarity, (b) constraints on prior coordination between speaker and listener, and (c) constraints on the resources available for inference. Further, we propose a simple generative model, under which there exists a computationally efficient algorithm with theoretical guarantees. Finally, we corroborate our results with an empirical study.

1 Introduction

Codenames [Chvátil, 2015] is a word reference game played in two teams, each made up of a *speaker* and a *listener*. Twenty-five words are drawn from a deck. Nine are assigned to the team that goes first, eight are assigned to team that goes second, seven are marked as neutral, and one is designated as the "assassin." All players see the words, but only the speakers see the partition assignments. The speakers take turns giving their partner a single *codeword* and integer k, and the listener has to guess which k words the speaker was thinking of, with the turn ending when the listener decides to stop guessing or guesses a word not belonging to their subset. The game ends when one team reveals all of the words, winning the game, or reveals the assassin and loses instantly.

Word reference games like Codenames provide a natural setting for studying the computational mechanisms underlying human word association. In this paper, we develop algorithms for playing Codenames, following from prior work in cognitive science and machine learning. In cognitive science, Shen et al. [2018] use a simplified version of the game to compare computational models of human similarity judgments and reference. In machine learning, Summerville et al. [2019] introduced the Codenames AI Competition, with subsequent work comparing heuristics for selecting codewords for a given set of target words, using word vectors or knowledge graphs [Kim et al., 2019, Koyyalagunta et al., 2021].

Following prior work, we focus on algorithms for generating codewords, assuming a speaker with some (possibly probabilistic) model of the listener's *similarity function*. First, we describe an algorithm for finding the codeword identifying the largest possible subset of target words for a given similarity function. Prior work enumerates over every possible subset of target words; our algorithm finds the maximum size subset in time linear to the vocabulary size. Second, we give a formal characterization of the optimal similarity structure in terms of a similarity matrix or a word association graph, and give bounds on the best expected performance given a codeword vocabulary with fixed size. We extend this analysis to a stochastic setting, where the speaker has a probabilistic model of the listener's similarity function. Third, we describe how similarity structures derived from real-world semantic resources differ from the optimal structure. Fourth, we develop an approximate search algorithm that improves the runtime of our search procedure, finding good codewords without looping over the vocabulary or enumerating all subsets of target words.

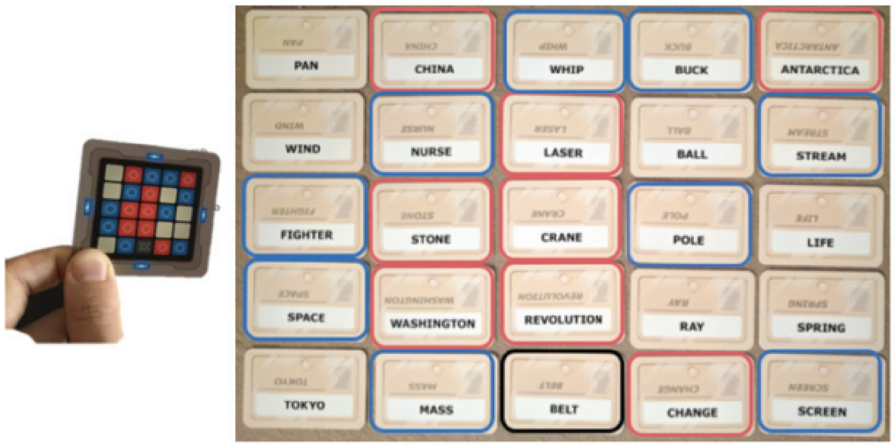

Figure 1: An example Codenames game board (right) and key (left). Each word is highlighted in the image (but not in the game) with the key's corresponding color.(Jaramillo et al. [2020])

We do simulations to illustrate the tradeoffs between different algorithms under different assumptions about the similarity structure and the computational resources available to the speaker. In a "bot-to-bot" setting, where both speaker and listener have access to the same word embeddings, our algorithm can consistently find clues that successfully pick out most or all of the target words in the first round, using only ten thousand 25-dimensional word vectors, which considerably outperforms the prior work. In preliminary user studies, we find that our algorithm finds human-guessable codewords for targets of size three or four, outperforming human and computer baselines.

2 Background

2.1 Notation

We will focus on a single turn of Codenames, which consists of a set of words $U \subset V$ and a set of target words $A \subset U$, where V is the vocabulary. Optionally, also partition $U \setminus A$ into opponent words B, neutral words C, and an "assassin" word $\{s\}$. Let S denote a speaker and L denote a listener. Both players can see U, but only S can see the partition assignments. In each turn, S picks a code word $c \in V$ and an integer k between 1 and $|A|$, and L responds by selecting words in U until either deciding to stop or revealing a word $w \notin A$. Let let k' denote the number of words guessed by L and let π denote the ordered sequence of k' words in U that L guessed. Then $\text{score}(\pi, k') = k' - 1 + \text{score}(w_{\pi_{k'}})$, where $\text{score}(w_{\pi_{k'}}) = 1$ if $w_{\pi_{k'}} \in A$, -1 if $w_{\pi_{k'}} \in B$, 0 if $w_{\pi_{k'}} \in C$, and $-\infty$ if $w_{\pi_{k'}} = s$. Our objective will be to find the clue (c, k) maximizing $\mathbb{E}[\text{score}(\pi, k')]$ under some model $\Pr(\pi, k' \mid c, k)$. We will also write π_c to denote the permutation of U induced by codeword c, with $\pi_{c,i}$ identifying the i^{th} word in the permutation, and assume for simplicity that $\Pr(\pi, k' \mid c, k) = \Pr(\pi \mid c)\delta(k - k')$, i.e. the listener always guesses k words.

2.2 Speaker/Listener Model

Following Shen et al. [2018], we use the Rational Speech Act model [RSA; Frank and Goodman, 2012] to define the extent to which speaker and listener recursively reason about the mental state of the other player. The RSA model consists of the following components:

- A literal listener L_0 calculates likelihoods of the form $\Pr_{L_0}(\pi, k' \mid c, k, \theta_{L_0})$, without recursively reasoning about the other player. We assume $\Pr_{L_0}$ is from a predefined class of probability functions with parameters θ_{L_0}.
- A pragmatic listener S_1 picks a cue (c, n) that maximizes the expected score, given some belief $P_{S_1}(\theta_{L_0})$ about the listener's likelihood function:

$$\mathbb{E}_{\Pr_{S_1}(\theta_{L_0})}\left[\mathbb{E}_{\Pr_{L_0}(W \mid c, n, \theta_{L_0})}[\text{score}(W)]\right]$$

- The true listener is either pragmatic (denoted L_1), and picks a subset by recursively reasoning about S_1, or literal (L_0), and picks the subset maximizing $\Pr_{L_0}(W \mid c, n, \theta_{L_0})$. We will mostly assume that the listener is literal.

The optimal strategy is for the speaker to pick the clue maximizing the expected score and for the listener to pick the maximum likelihood subset. In the remainder of this paper, we explore questions that arise in implementing this strategy, including: what is the form of the likelihood function, $\Pr_{L_0}(W \mid c, n, \theta_{L_0})$; how much knowledge does S_1 have about $\Pr_{L_0}$ (and to what extent can the players coordinate); and how can the players approximately identify good clues and subsets if we impose restrictions on the computational resources available for inference.

2.3 Baselines

Prior work in machine learning adopts some form of the following algorithm:

- The listener uses a **factored likelihood** function:

$$\Pr_{L_0}(W \mid c, n, \theta_{L_0}) = \mathbb{1}\{|W| \leq n\} \prod_{w \in W} \Pr_{L_0}(w \mid c, \theta_{L_0}),$$

  where $\mathbb{1}$ is the indicator function and $\Pr_{L_0}(w \mid c, \theta_{L_0}) \propto \text{sim}_{\theta_{L_0}}(c \to w)$ is proportional to pairwise similarities defined by θ_{L_0}.
- The speaker **enumerates subsets**: for each k, the speaker enumerates all subsets $W \subseteq A$ of size k and then loops over the vocabulary to find the codeword c such that $\min_{w \in W} \text{sim}(c \to w) > \max_{u \in U \setminus W} \sim (c \to u)$ and $\min_{w \in W} \text{sim}(c \to w) > t$ for a heuristically chosen threshold t.

The main difference between prior work is then in the choice of how to parameterize $\text{sim}(c \to w)$, for example using word vectors or a knowledge graph [e.g. Kim et al., 2019, Koyyalagunta et al., 2021]. In this work, we will adopt the first assumption (factored likelihood). However, enumerating subsets is clearly sub-optimal as it incurs a cost of at least $O(|A|!|V||U|)$ to enumerate over every subset of target words in A, then over the vocabulary, and then over all of the game words U to compare pairwise distances.

3 Algorithms

In this section, we develop algorithms and analysis for Codenames given a series of assumptions about the similarity function; the speaker's knowledge of the similarity function; and computational constraints. First we describe a similarity function for a deterministic protocol, along with an $O(|A||V|)$ search function, that finds the maximum scoring subset, and present an urn-problem construction for analyzing the trade-off between expected score and vocabulary size. Then we extend the analysis to a stochastic setting, where the speaker has a noisy probabilistic model of the listener's similarity function. Third, we present an approximate-nearest-neighbors algorithm that reduces the cost of our search algorithm to $O(|A|\sqrt{|V|})$ [Dan: confirm]
.

3.1 Deterministic protocol

In the deterministic protocol, we assume that $P(W \mid c) = \prod_{w \in W} P(w \mid c)$, $P(w \mid c) \propto \text{sim}(c \to w)$, and the speaker and listener have the same similarity function $\text{sim}(c \to w)$. The main limitation in this setting is on the size of the vocabulary and the structure of the similarity function. (Otherwise, the first team could guess all words on their first turn by assigning every possible subset of $|A|$ game words to a single unique word in the vocabulary.)

3.1.1 Deterministic clue finder

Observation Our main observation is that for any similarity function $\text{sim}(c \to w)$, every codeword c corresponds to a permutation of the vocabulary, π_c, induced by sorting the vocabulary in decreasing order of $\text{sim}(c \to w)$ with ties broken arbitrarily. We can then define the score of a codeword, $\text{score}(c) = \text{score}(\pi_c)$, where $\text{score}(\pi_c)$ is equal to the maximum index i such that $\pi_{c,j} \in A$ for all $j \le i$. Clearly, the optimal deterministic strategy is to select the clue c maximizing $\text{score}(\pi_c)$.

Algorithm 1 Deterministic clue-finder

```
Require: a vocabulary $V$, game words $U$, a similarity function $\text{sim}(c \to w)$ for all $c, w \in V$.
  Preprocess to get $\pi_{c,i}$, the word with the $i^{th}$ largest value of $\text{sim}(c \to w)$, for $i \in \{1, \ldots, |V|\}$.
  for $c \in V$ do
      Let $k_c$ be the smallest index such that $\pi_{c,i} \in A$ for all $i \le k_c$.
  end for
  Let $\hat{c} = \arg\max_c k_c$.
  Return $\hat{c}, k_{\hat{c}}$.
```

Analysis of Algorithm 1. Algorithm 1 finds a maximum scoring subset by construction. The version described above using $O(|V| \times |V|)$ memory to precompute π_c for all c and runs in time $O(|U||V|)$. For example, if we precompute a map from every w to its rank in π_c, we can find k_c by finding the minimum-rank word in $U \setminus A$ by enumeration. We could trade off memory for time by calculating $\text{sim}(c \to w)$ on the fly for all $w \in U$ and the sorting, resulting in time $O(|V||U| \log |U|)$. Either solution is an improvement over the baselines (Section 2.3), which run in time $O(|A|!|V||U|)$.

3.1.2 Optimal deterministic similarity function

Now we consider the properties of an optimal similarity function, assuming that speaker and listener can communicate prior to the game.

Lemma 1. *Suppose $c \in V$ is a vocabulary word; π_c is drawn uniformly at random from the set of all permutation of indices into V; $U \subset V$ is a set of randomly drawn game words with target words $A \subset U$; and $\text{score}(\pi_c)$ is the maximum index j such that $\pi_{c,i} \in A$ for all $i \le j$. Then the probability that the score is greater than or equal to some integer k is:*

$$\Pr\left(\text{score}(\pi_c) \ge k\right) = \frac{\binom{|A|}{k}}{\binom{|U|}{k}}.$$

Proof. This problem is an instance of the urn problem without replacement, which defines the hypergeometric distribution. Consider an urn containing $|U|$ marbles, with $|A|$ colored blue and the rest colored red. Each code word defines a random ordering of the vocabulary—and so also a random ordering of U—which can be modeled as a sequence of draws from U without replacement. The probability of achieving score $\ge k$ is equal to the probability of drawing k blue marbles in k turns, which equal to the number of subsets of k elements in A divided by the number of subsets of k elements in U. □

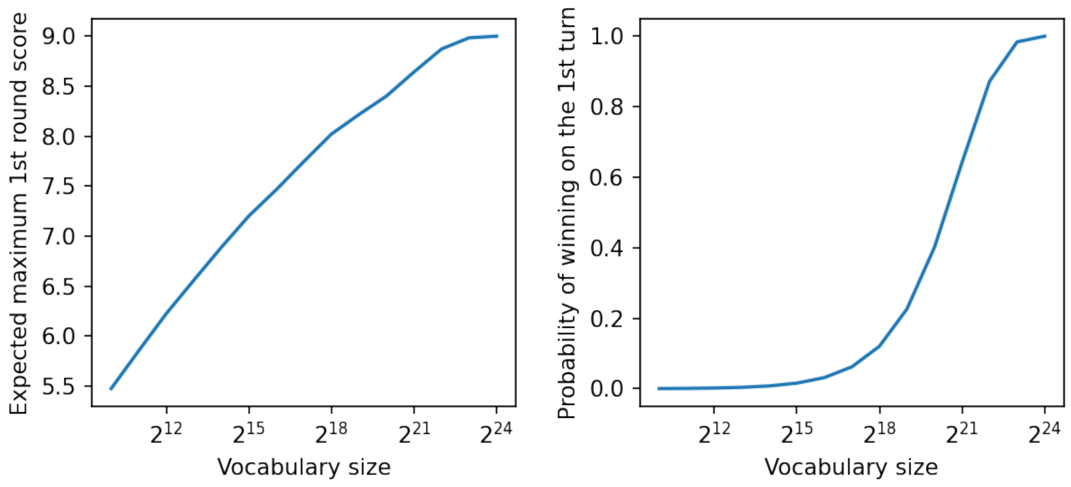

Figure 2: The expected value of the maximum scoring clue for a random game (left) and the probability of guessing all nine words on the first turn of a random game (right) as a function of vocabulary size, assuming word similarities are drawn uniformly at random from the set of permutations.

Now we can characterize the relationship between the size of the vocabulary and the expected performance of the deterministic protocol.

Theorem 3.1. *Suppose that, for all $c \in V$, π_c is drawn uniformly at random from the set of all permutation of indices into V and fixed; $U \subset V$ is a game; and $A \subset U$ is the set of target words. Then the probability that there is a codeword $c \in V$ with $\text{score}(c) \geq k$ is*

$$\Pr(\exists c \in V : \text{score}(c) \geq k) = 1 - \left(1 - \frac{\binom{|A|}{k}}{\binom{|U|}{k}}\right)^{|V|}.$$

Proof. The probability that every codeword has score less than k is equal to the product of the probabilities that each individual word in $|V|$ has score less than k, which is given by the lemma. The probability that any codeword has score $\geq k$ is the remaining probability. □

Discussion We plot how the probability of finding a codeword changes with $|V|$ and k in Figure 2. In the first round of Codenames, $|U| = 25$ and $|A| = 9$. Assuming that π_c is a random and shared between speaker and listener, even with a vocabulary containing only $2^{12} = 4096$ words, we can expect to find clues that identify more than six target words on the first round, which, anecdotally, is much better than human performance. With ten million words we can expect to guess all of the target words on the first turn more than 99% of the time. We simulate this scenario in Section 4. In the next section, we extend this analysis to the stochastic setting where the speaker does not know the listener's similarity function exactly, and then to a scenario where the π_c are not uniformly random permutations but are instead derived from a pre-existing word similarity graph.

Open question: are random similarities optimal? For a fixed vocabulary size, is there a similarity function $\text{sim}(c \to w)$ that achieves better expected score than the random permutations described above? We leave this question for future work, but offer an intuitive proof sketch here. For fixed integer k, every permutation π_c can also be thought of as describing a distribution over games, $\Pr(G = U, A \mid \pi_c)$, which is uniform over all of the games such that k words in A appear before any words in U in π_c, and equal to 0 everywhere else. The expected performance of a set of permutations $\Pi = \{\pi_c\}_{c \in V}$ can be expressed in terms of the number of games $G = U, A$ for which the marginal probability $\Pr(G \mid \Pi) = \frac{1}{|V|} \sum_{c \in V} \Pr(G \mid \pi_c)$ is non-zero. This term resembles a divergence between $\Pr(G)$ and $\Pr(G \mid \Pi)$, where $\Pr(G)$ is the probability of drawing a particular game. $\Pr(G)$ is uniform over games, hence $\Pr(G \mid \Pi)$ ought to be as close as possible to uniform, i.e. the maximum entropy distribution. This proof sketch indicates that a random similarity function might in fact be optimal.

3.2 Stochastic protocols

In the previous section, we made the assumption that speaker and listener share the same similarity function. In this section, we extend our analysis to a more realistic setting in which the speaker has only a noisy probabilistic model of $\text{sim}(c \to w)$.

3.2.1 Distributions over rankings

In order to calculate probabilities under the framework from Section 3.1, we need a function that can assign probabilities to rankings. Here we construct such a distribution by extending the urn construction of the hypergeometric distribution.

Categorical urn construction ("exploded logit" distribution) Consider an urn containing N marbles and let $\Pr(x_i) = \mathbf{p}_i$ denote the probability that marble i is the next marble drawn from the urn, which is given by categorical distribution with parameter $\mathbf{p}$. Then the probability of a ranking π of elements in N is the probability of drawing the elements in π in order:

$$\Pr(\pi) = \prod_{i=1}^{N} \Pr(x_{\pi_i} \mid x_{\pi_{<i}}),$$

where $\Pr(x_{\pi_i} \mid \pi_{<i})$ is the categorical distribution obtained by setting $\Pr(x_j) = 0$ for all $j \in \pi_{<i}$ and renormalizing π. Thus we have a distribution over rankings, which we can parameterize with multinomial parameters $\mathbf{p}_c$ for each $c \in V$. This distribution is also called the "exploded logit" distribution [Nicenboim, 2021] and has been used to model the outcomes of horse races [Gakis et al., 2018].

Two-color categorical urn construction In order to use the categorical urn construction to model Codenames, we need to augment it by coloring the marbles. Recall that the main value we need to compute is the probability that the first k elements in a permutation π are all drawn from A. If π is unknown, we would like to calculated the expectation:

$$\mathbb{E}_{\Pr(\pi)}[\mathbb{1}\{\text{score}(\pi) \geq k\}] = \sum_{\pi} \Pr(\pi)\mathbb{1}\{\text{score}(\pi) \geq k\} = \sum_{\pi' \in \text{perm}(A,k)} \Pr(\pi').$$

Here, $\text{perm}(A, k)$ denotes the set of all permutations of k elements in W. $\Pr(\pi')$ is the probability of drawing this permutation before drawing any other elements of U. While the naive solution requires enumerating over every permutation of U, we only need to calculate the sum of the probability of every permutation of k elements in A, which gives the probability that the first k elements drawn from the urn are all from A. Still, enumerating every permutation of k elements imposes an unwanted cost of $O(|A|!)$.

Approximate inference for the two-color categorical urn Unfortunately, we have not yet found an efficient closed form expression to calculate $\mathbb{E}_{\Pr(\pi)}[\text{score}(\pi)]$ without marginalizing over permutations, so we consider two simple schemes for approximate inference.

1. **Monte Carlo estimate:** Given categorical distributions $\Pr(w \mid \mathbf{p}_c)$ for each w, c, we can sample from $\Pr(\pi_c \mid \mathbf{p}_c)$ in $O(|A|)$ time by sampling words without replacement until sampling a word in $U \setminus A$.
2. **Maximum likelihood estimate:** We can take a lower bound by using the maximum likelihood point estimate:
$$\mathbb{E}[\text{score}(\pi)] = \sum_{\pi} \Pr(\pi)\text{score}(\pi) \geq \Pr(\pi^*)\text{score}(\pi^*),$$

where π^* is the maximum likelihood ordering obtained by selecting the marble i with maximum probability $\mathbf{p}_i$ at each time step without replacement. This model is now equivalent to the deterministic model from above with the difference that the expected score is now weighted by $\Pr(\pi^*)$.

Stochastic speaker/listener model If both speaker and listener have access to the same categorical parameters $\mathbf{p}$, then they can recover the deterministic protocol by both taking maximum likelihood estimates. In order to make the setting stochastic, we can require that either the listener or both speaker and listener can only take a finite number of samples from the distribution. Alternatively, we can assume that speaker and listener have different categorical parameters, $\mathbf{p}_c^S$ and $\mathbf{p}_c^L$, bot hsampled from a shared prior, Dirichlet(α_c), where α_c is the concentration parameter. We call this model the **Two-color Dirichlet urn construction**. As we will discuss in the next section, using a Dirichlet prior over word association strengths also gives us a natural way of modeling the effect of the types of sparse structure that appear in real world word similarity graph. (Note that an optimal speaker would estimate the Dirichlet parameters; we leave this exploration for future work and assume that the speaker models $\Pr(\pi_c)$ using only point estimates or samples from their own similarity distribution.)

3.3 Approximate nearest neighbor search

To reduce the complexity of search of a clue-word for the speaker, we can follow the following strategy:

1. Create k clusters over words in the vocabulary V. Denote the clusters by $\mathcal{C}_1, \dots \mathcal{C}_k$ and their centers as $v_{C_1}, \dots, v_{C_k}$ respectively.
2. Given $U = A \cup B$, search the cluster center v_{C_i} that maximizes the following score:

$$\begin{aligned}&\max_{W \subset A} |W| \\ &\text{s.t. for all } w \in W, dist(w, v_{C_i}) < dist(b, v_{C_i}) \text{ for all } b \in B.\end{aligned}$$

3. Search for the clue-word c only in the cluster C_i.

Theorem 3.2. *Under the assumption that the k clusters are uniform in size, the above algorithm computes a clue-word in $\mathcal{O}(k + \frac{|V|}{k})$ operations. Thus, under the same assumption, the number of operations can be reduced to $\mathcal{O}(\sqrt{|V|})$, when $k = \sqrt{|V|}$.*

Proof. The algorithm searches the clue-word in two steps:

1. First, it searches over the k-cluster centers to find the best cluster to focus on. This involves $\mathcal{O}(k)$ operations.
2. Second, it searches over the words in a specific cluster found in the first step. Under the assumption that all clusters are uniform in size, the total number of words to search over in the cluster becomes $\frac{|V|}{k}$, and hence the total operations involved can be bounded by $\mathcal{O}(\frac{|V|}{k})$.

$\square$

3.4 A provable algorithm under a generative model

To show the existence of an efficient algorithm, that can choose the optimal clue-word without having to go over the entire vocabulary, we need to take some structural assumptions on the word embeddings. We assume that there exist a set of concepts/topics underlying the word embeddings. A word's embedding implicitly contains the relevance of the topics for the word. We are interested in the setting, where given

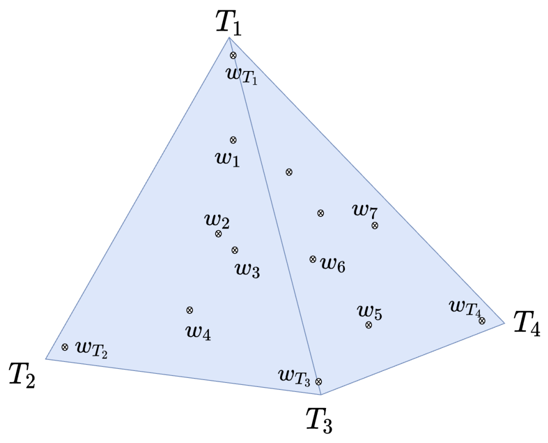

Figure 3: A figurative description of the generative model. The vertices of the simplex are given by the topics $T_1, \dots, T_k$, and the words are present inside the simplex. Each topic has at-least one anchor word.

words $U = A \cup B$, the algorithm has to return a clue-word c, such that c differentiates as many words in A as possible from the words in B, based on at-least one concept.

A running example can be given as follows: suppose there are two topics, sport and animal. A word like cat can be represented by the embedding for the topic animal, while the word bat can be represented by a convex combination of the embeddings of the two topics. To differentiate between $\{\text{dog}, \text{cat}\}$ and $\{\text{soccer}, \text{baseball}\}$, one can use the concept of sports, and hence any word representing sports suffices.

Formally, Consider the following generative model underlying the words in the vocabulary V:

1. There exist a set of topics $T_1, \dots, T_k$. Each topic T_i is represented by a vector $v_{T_i} \in \mathbb{R}^d$. Let $T \in \mathbb{R}^{d \times k}$ denote the matrix, where each column corresponds to a unique topic vector.
2. Each word w is generated as follows:
   (a) Sample $p_w \in \Delta^k$, where Δ^k denotes the space of 1-simplex in $\mathbb{R}^k$.
   (b) Generate $v_w = Tp_w$.

A figurative explanation is given in fig. 3.

For the provable efficient algorithm that we propose based on the generative model, we need to take an assumption:

Assumption 3.3 (Anchor word assumption)**.** For each topic T_i, there exists an anchor word $w_{T_i} \in V$ that specifically belongs to the topic T_i, i.e. $p_{w_{T_i}}[i] = 1$ and $p_{w_{T_i}}[j] = 0$ for all $j \in [k] \setminus \{i\}$.

We take the above assumption for the simplicity of presentation. The above assumption can be strengthened to the case where for each topic T_i, there exists an anchor word s.t. $p(T_i|w) > p$ for some constant p.

3.4.1 Efficient provable algorithm

The main idea is to capture the anchor words $\{w_{T_i}\}_{i \in [k]}$ in the vocabulary. A simple observation from the generative model is that the words lie in a k-dimensional simplex in $\mathbb{R}^d$, with the anchor words forming the vertices of the simplex. Thus, the main idea can be summarized as capturing the vertices of the convex hull of the word embeddings $\{v_w\}_{w \in V}$. The algorithm to find the anchor words has been summarized in figure 2. We have taken the algorithm from Arora et al. [2013].

Once we know the k anchor words in the vocabulary, the problem of searching a clue-word boils down to finding the anchor word that is closer to a subset of words in A, compared to the words in B. To find such a subset, we only need to look at each anchor word, sort the words in U according to their distance to the anchor word, and select the anchor word that maximizes the number of words in A closer to itself than the words in B. The algorithm has been summarized in figure 3.

Algorithm 2 Anchor word locator (Arora et al. [2013])

```
Require: a set of vertices $\{v_w\}_{w \in V}$ in $d$-dimensions, present in a simplex with $K$ vertices, and some constant $\epsilon > 0$.
  $S \leftarrow \{v_w\}$ s.t. $v_w$ is the farthest point from the origin.
  for $i = 1, \ldots, k-1$ do
      Let $v_{w'}$ be the point in $\{v_w\}_{w \in V}$ that has the largest distance to $span(S)$.
      $S \leftarrow S \cup \{v_{w'}\}$.
  end for
  Let the current $S = \{v'_1, \ldots, v'_K\}$.
  for $i = 1, \ldots, K$ do
      Let $v_{w'}$ be the point in $\{v_w\}_{w \in V}$ that has the largest distance to $span(S) \setminus \{v'_i\}$.
      Update $v'_i$ to $v_{w'}$.
  end for
  Return $S$.
```

Algorithm 3 Clue-word finder

```
Require: Codename words $U$, partitions $A$ and $B$, Candidate clue-words $S$.
  $D \leftarrow []$.
  for $c \in S$ do
      Set $D[c] = |\{a \in A \mid \text{dist}(a, c) < \min_{b \in B} \text{dist}(b, c)\}|$.
  end for
  Return $\arg\max_{c \in S} D[c]$.
```

3.4.2 Theoretical guarantees

Theorem 3.4 (Arora et al. [2013])**.** *Algorithm 2 runs in time $\mathcal{O}(|V|^2 + d\,|V|\,k)$ and returns the vertices of the convex hull P of the words $\{v_w\}$ in the vocabulary.*

Theorem 3.5. *If S denotes the set of anchor words in the vocabulary, then algo 3 finds the optimal clue-word in time $\mathcal{O}(k)$ w.r.t. the given set of clue-words:*

$$\{c \mid \exists T_i \in \{T_1, \ldots, T_k\} \text{ s.t. } \text{ordering}(U, c) = \text{ordering}(U, T_i)\},$$

where $\text{ordering}(U, x)$ *is the sorted order of the set U w.r.t. the distance of the embeddings of the words in U with the embedding of x.*

Proof. By assumption 3.3, for each topic T_i, its corresponding anchor word w_{T_i} has the same embedding as v_{T_i}. Since algo 3 searches over all possible orderings of the set U w.r.t. the anchor words, implicitly it searches over the possible orderings w.r.t. the topics $\{T_i\}_{i=1}^k$. □

Thus, the above algorithm works, when we search over clue-words that differentiate words in U on the basis of at-least one concept/topic. Improving the generative model to have theoretical guarantees w.r.t a more stronger class of possible clue-words remains an interesting future direction.

4 Experiments

4.1 Setup

We simulate our algorithms by drawing 1000 random games from the standard Codewords vocabulary and testing speakers and listeners using different similarity functions. We construct similarity functions from two pre-existing semantic resources, along with random similarity functions.

Small World of Words Small World of Words (SWOW; De Deyne et al., 2019) is a semantic association graph that was collected by showing human subjects a series of cue words and asking them to write down the first three words that came to mind. We create a $|V| \times |V|$ matrix S, with $S_{i,j}$ equal to the number of participants who responded with word j in response to word i. ($|V| = 12K$ and $\max_{i,j}(S_{i,j}) = 98$). We set $\text{sim}(c \to w) = \text{sim}(w \to c) = S_{i,j} + S_{j,i}$, which assumes that the similarity function is symmetric.

GloVe-Twitter-25d GloVe vectors [Pennington et al., 2014] are real-valued vectors created by approximately factorizing a matrix of word co-occurrence counts. We use 25-dimensional vectors that were trained on 27-billion tokens from Twitter, which we download from the gensim library [Rehurek et al., 2011]. We normalize the vectors and set $\text{sim}(c \to w) = \mathbf{v}_c^\top \mathbf{v}_w$, where $\mathbf{v}_c$ and $\mathbf{v}_w$ are the vectors for c and w.

Random similarities We also use two random similarity functions. Random permutations are created by sampling $|V|$ permutations of of the vocabulary uniformly at random, and setting $\text{sim}(c \to w)$ to be proportional to the position of w in the permutation assigned to c. Random vectors are 25 dimension vectors with entries drawn independently from the standard normal distribution, and $\text{sim}(c \to w) = \mathbf{v}_c^\top \mathbf{v}_w$.

4.2 Code-finding experiments

	Deterministic			Random		
sim function	1st-turn score	1st-turn advantage	# rounds	1st-turn score	1st-turn advantage	# rounds
Random permutations	6.7	0.822	1.816			
Random vectors 25d	6.3	0.858	1.863			
Small World of Words	3.6	0.573	3.164			
GloVe-Twitter-25d	5.7	0.667	1.998			

Table 1: Results averaged over 1000 simulations of Codenames. $|V| = 11,808$ for all models, which is the number of words in the intersection of the Small World of Words and GloVe vocabularies. **1-st turn score** is the average number of words guessed correctly after the first codeword. **1-st turn advantage** is the proportion of games won by the team that went first. **# rounds** is the average number of rounds to end the game. See Section 4.1 for more details about the similarity functions.

Deterministic setting The first columns of Table 1 compares the performance of speakers and listeners with different similarity functions using the deterministic protocol in Section 3.1. In these settings, we assume that speaker and listener have identical, deterministic similarity functions. The best performance is achieved by the random permutations model, which assigns every codeword to a random permutation of the vocabulary. The average first turn score attained by this model is 6.7, which is the same as the expected first-turn score predicted by Theorem 3.1 for $|V| = 11,808$ (6.73). Random 25-dimensional vectors perform worse than random discrete permutations but better than GloVe vectors. Overall, these results support our conjecture in Section 3.1 that the best similarity function for codenames is the one with the highest entropy over permutations of the vocabulary.

The semantic association graph performs the worst, for reasons we discuss in more detail in the next section.

Stochastic setting Unfortunately we could not implement the stochastic setting in time, so we will leave it for future work.

4.3 ANN experiments

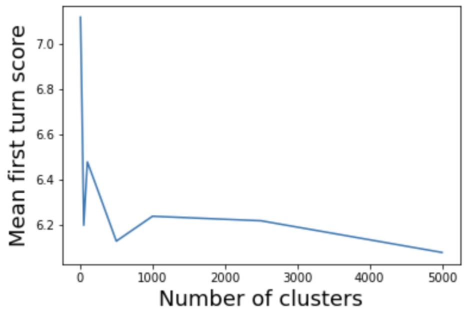

Figure 4: Average number of words predicted with first clue

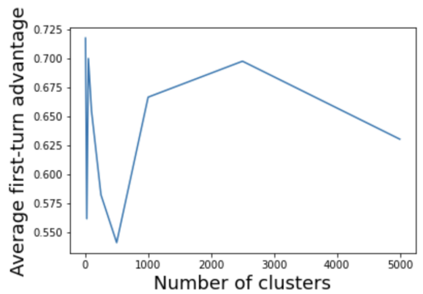

Figure 5: Average first-turn advantage.

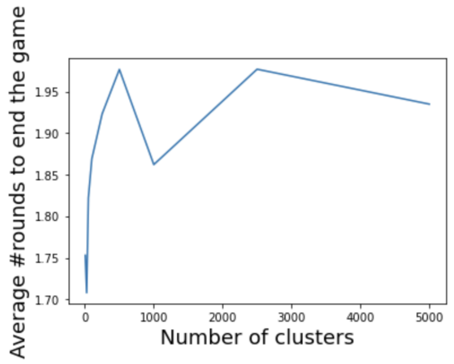

Figure 6: Average number of rounds to complete the game.

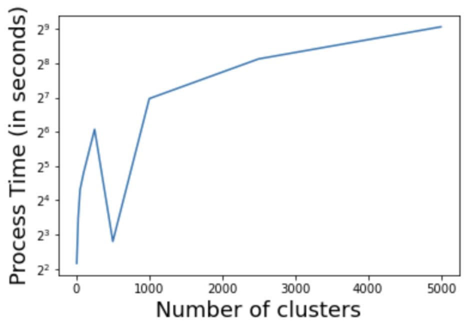

Figure 7: Time to compute Clusters

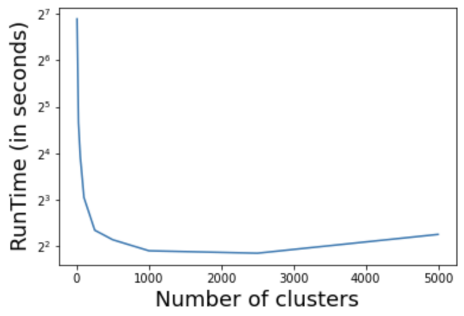

Figure 8: Time to compute clues for 100 games

We perform experiments on the approximate nearest neighbor search algorithm, in the setting where the vocabulary size is large. For the experiments, we consider the setting where the clue-word can come from the GloVe vocabulary, where $|V| = 1193514$.

We use the standard kmeans clustering (Sculley [2010]) to cluster the GloVe emebddings of the words in our vocabulary into k clusters. Then, we follow the algorithm outlined in section 3.3 to compute the clue-word at the speaker.

Fig. 4 shows the behavior of the number of words hit with the first clue, averaged over 100 games, with respect to the number of clusters k. In Fig. 5, we show the behavior of the the $1st$-turn advantage, which computes the proportion of the games won by the first team that went first, with the number of clusters k. In Fig. 6, we show the behavior of the average number of rounds necessary to end the game. In fig. 7 and fig. 8, we plot the computation time involved in the clustering algorithm and the average time to compute the first clue-word, and their behavior with k respectively. All the plots have been averaged over 100 simulations.

The observations can be succinctly summarized as follows:

1. With increase in the number of clusters, the average number of words hit with the first clue-word drops mildly. This was expected, since we increasingly localize the search of a clue-word, with increasing number of clusters.
2. The average number of rounds taken by the 2 teams generally increases with increase in the number of clusters. The first-turn advantage also seems to decrease with the increasing number of clusters, although we don't see a clear trend as other plots. The increase in the number of rounds can be explained in terms of the drop in the number of codenames recovered in each turn, as observed before. However, here we have only assumed that the players are single-minded, i.e. they only try to increase their chances of winning, without trying to hurt the other team. Calculating the performance of the approximate algorithm in a competitive game framework remains an interesting problem.
3. The pre-processing time, which involves running the kmeans algorithm to find k-clusters, increases with increasing number of clusters.
4. The run-time to compute a clue-word for 100 different games drastically decreases with increasing number of clusters, since we are further localizing the search with the increase in k.

5 Discussion

In this section, we give an example of a board-game and show the codenames returned by the different similarity functions (see section 4 the details). Here, we consider the case where the clue-word can come only from the set of words present in the vocabularies of both GloVe and SWOW.

The set of words on the board are given by

'organ', 'teacher', 'school', 'spring', 'sock', 'poison', 'superhero', 'plate', 'change', 'state', 'jam', 'soldier', 'temple', 'ship', 'code', 'plot', 'screen', 'litter', 'brush', 'shark', 'pyramid', 'compound', 'hand', 'mexico', 'press'.

and the random partition is given by

'A': 'change', 'organ', 'plate', 'poison', 'school', 'sock', 'spring', 'superhero', 'teacher',
'B': 'soldier', 'temple', 'code', 'jam', 'plot', 'state', 'ship', 'screen',
'Neutral': 'shark', 'brush', 'mexico', 'litter', 'compound', 'pyramid', 'hand',
'Assassin': 'press'.

Here, $'A'$ and $'B'$ denote the set of words belonging to teams A and B, while $'Neutral'$ and $'Assassin'$ denote the set of neutral words and the assassin word respectively.

On running the deterministic algorithm on the basis of GloVe embeddings, we get a clue-word for 6 intended code-words, with the clue 'romantic' in the first turn for the speaker in team A, and the intended set of code-words for the listener in team A are superhero', 'change', 'spring', 'organ', 'poison', 'teacher'.

However, On running the deterministic algorithm on the basis of SWOW similarity matrix, we get a clue-word for 4 intended code-words, with the clue 'clarify' in the first turn for the speaker in team A, and the intended set of code-words for the listener in team A are

'teacher', 'school', 'spring', 'change'.

This illustrates that the underlying structure of the similarity matrix changes, with different datasets. Although clue-word derived based on GloVe gives more intended code-names in this example, the clue-word derived based on SWOW seems to be more semantically related to the intended code-names. Understanding the generative model that returns interpretable clue-words and also returns optimal number of intended code-names remains an interesting open problem.

6 Conclusion

In this paper, we explored algorithms for the popular reference game Codenames. We describe the optimal solution to the computational problem, and then develop a range of algorithms for approximately solving the problem under different resource constraints, including (a) constraints on the data structures and primitives for calculating word similarity, (b) constraints on prior coordination between speaker and listener, and (c) constraints on the resources available for inference. Further, we propose a simple generative model, under which there exists a computationally efficient algorithm with theoretical guarantees. Finally, we corroborate our results with an empirical study.

Acknowledgements

References

Sanjeev Arora, Rong Ge, Yonatan Halpern, David Mimno, Ankur Moitra, David Sontag, Yichen Wu, and Michael Zhu. A practical algorithm for topic modeling with provable guarantees. In *International conference on machine learning*, pages 280–288. PMLR, 2013.

Vladimír Chvátil. Codenames, 2015.

Simon De Deyne, Danielle J Navarro, Amy Perfors, Marc Brysbaert, and Gert Storms. The "small world of words" english word association norms for over 12,000 cue words. *Behavior research methods*, 51(3): 987–1006, 2019.

Michael C Frank and Noah D Goodman. Predicting pragmatic reasoning in language games. *Science*, 336 (6084):998–998, 2012.

Konstantinos Gakis, Panos Pardalos, Chang-Hwan Choi, Jae-Hyeon Park, and Jiwun Yoon. Simulation of a probabilistic model for multi-contestant races. *Athens Journal of Sports*, 5:95–114, 2018.

Catalina Jaramillo, Megan Charity, Rodrigo Canaan, and Julian Togelius. Word autobots: Using transformers for word association in the game codenames. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 231–237, 2020.

Andrew Kim, Maxim Ruzmaykin, Aaron Truong, and Adam Summerville. Cooperation and codenames: Understanding natural language processing via codenames. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 160–166, 2019.

Divya Koyyalagunta, Anna Sun, Rachel Lea Draelos, and Cynthia Rudin. Playing codenames with language graphs and word embeddings. *Journal of Artificial Intelligence Research*, 71:319–346, 2021.

Bruno Nicenboim. A simple way to model rankings with stan. https://bnicenboim.github.io/2021/03/21/a-simple-way-to-model-rankings-with-stan/, 2021. [Online; accessed 2021-12-14].

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Radim Rehurek, Petr Sojka, et al. Gensim—statistical semantics in python, 2011.

David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.

Judy Hanwen Shen, Matthias Hofer, Bjarke Felbo, and Roger Levy. Comparing models of associative meaning: An empirical investigation of reference in simple language games. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 292–301, 2018.

Adam Summerville, Andrew Kim, Maxim Ruzmaykin, and Aaron Truong. The Codenames AI Competetion. In *FDG '19: Proceedings of the 14th International Conference on the Foundations of Digital Games*. Association for Computing Machinery, 2019.