Information Theory in a Card-Guessing Game

Albert Gural, Mackenzie Simper, Ernest So

March 24, 2019

Abstract

In this work, we explore tools in information theory to analyze a card guessing with feedback. Consider a card guessing game where the player knows the initial composition of a shuffled deck (e.g. 3 Jacks, 3 Queens, 3 Kings) and guesses the next card in the deck with the goal of maximizing the number of correct guesses until no cards remain. A dealer gives the player feedback on their guess, which may give the player information to use for the following guesses. We focus on incomplete feedback where the player is told whether their guess was correct. In this scenario, the player's guess influences both the amount of information they know about the deck and the number of correct guesses. We will explore different strategies for this game and information theoretic ways to analyze it.

1 Introduction

When presented with a deck of cards a natural guessing game arises: A dealer turns over the cards one at at time and at each step the player guesses the value of the card. Supposing the deck is uniformly randomly shuffled, what is the guessing strategy that optimizes the expected number of correct guesses?

The optimal strategy depends on the type of *feedback* the player receives after each guess. In the *complete feedback* setting, the card at each step is revealed to the player. Thus at every step the player has complete information on the cards that have already been passed. More challenging, the dealer could simply tell the player if the guess was correct or incorrect; if the guess is incorrect, the true value of the card is not revealed. It is this *partial feedback* game that is the focus of this project.

Because the information that a player has at each step of the game is dependent on her past guessing sequence, it is difficult to directly compare strategies. A natural first guess at an optimal strategy is to be "greedy": Conditional on the past information, choose the card that is most likely to remain in the deck. When there are no repeated types of cards in the deck (for example, when asked to guess both suit and numerical value of cards in a standard deck), this greedy strategy is indeed optimal. However, when there are repeated cards of the same type (for example, when asked to guess the suit of cards in a standard deck), the greedy strategy may not be optimal. This surprising result was noted in [1]. Except for the case where no types are repeated, no simple description of the optimal strategy is known. The only characterization comes from a recursion that the expected number of correct cards by using the optimal strategy must satisfy (see Section 2.1).

The intuition is that sometimes choosing a card that is less likely to be correct will give the player more information about the deck, and this additional information will result in more correct guesses in the future. Of course, it is not immediately obvious what is the correct measure of

"information" gained from a guess. We choose to study the entropy of the remaining deck. That is, assuming you know the probability distribution of the deck at the beginning of guessing, your sequence of guesses and feedback gives you a new probability distribution on the cards that remain in the deck. We can calculate the entropy of this distribution, conditional on the information that the next guess provides. Since entropy is a measure of the "randomness" of a probability distribution, the natural idea is to choose the card that *minimizes* the entropy.

When there are no repeated cards in the deck, explicit calculations suggest that minimizing the entropy does recover the optimal strategy. For more complicated decks, however, minimizing entropy will not be optimal, and will not even do as well as greedy. It is not clear what is the optimal trade-off between minimizing entropy and maximizing the probability the next guess is correct. Theoretical proofs are complicated, and so we explore this question numerically in Section 5.

1.1 Outline

In Section 2 we discuss the background from [1] on the partial feedback guessing game. Recursions that define the optimal strategy are given. Then we introduce entropy-minimizing strategies, assuming the reader has basic background in information measures. In Section 2.3 we work through an example where the optimal strategy is *not* the greedy strategy, but minimizing entropy does give the optimal result. In Section3 we discuss the special case where the deck has no repeated cards of a single type; though it is known that the optimal strategy in this case *is* greedy, the simplicity allows us to actually write down some entropies. In Section 4, some information theoretic bounds for the *complete feedback* game are given. In Section 5 we discuss code written to numerically simulate the optimal strategy, and results testing different strategies. In Section 2 we discuss other strategies that are feasible to implement in real-life. Finally, in Section 7 we summarize the outreach event and finish with conclusions.

2 Set-Up

In this section, we set-up the notation necessary to discuss strategies in the card-guessing game. In Section 2.2 we introduce minimizing entropy strategies and in Section 2.3 we work through an example in which the greedy strategy does not give the optimal guess, but minimizing entropy does.

For *labels* of cards we use letters $a, b, c, \ldots$ and for the *position* of the card in the deck we use numbers $1, \ldots, N$. Let $\mathbf{c} = (c_1, \ldots, c_r)$ be the initial "composition" vector for a deck of N cards, i.e. there are c_1 cards of type 1, c_2 cards of type 2, ... so that $\sum_{i=1}^{r} c_i = N$. Let $\mathbf{c}_j$ be the *known* composition vector of the deck after j cards have been guessed. If the deck is uniform random, a key observation is that, before step $j+1$, the *position* of the "no's" for card a does not affect the probability that a is at position j. Since only the *number* of "no's" is important, we can store a vector $\mathbf{p}_i = (p_1, \ldots, p_r)$, where p_a is the number of "no's" received when guessing card a. Then the "information" gained from the guessing sequence after j guesses is determined by $(\mathbf{c}_j, \mathbf{p}_j)$.

2.1 Optimal Strategy Recursions

We can now write down a recursion that the expected number of correct guesses using the optimal strategy must satisfy. For $1 \leq a \leq r$, let $\overline{\delta}_a$ be the vector with a 1 at position a and 0s everywhere else. Then if at step j you guess card a, the vectors will update:

- If the guess is correct, then $\mathbf{c}_{j+1} = mc_j - \overline{\delta}_a$ and $\mathbf{p}_{j+1} = \mathbf{p}_j$.

- If the guess is incorrect, then $\mathbf{c}_{j+1} = \mathbf{c}_j$ and $\mathbf{p}_{j+1} = \mathbf{p}_j + \overline{\delta}_a$.

Let $P(a; \mathbf{c}_j, \mathbf{p}_j)$ be the probability that card a is at position j, conditional on $\mathbf{c}_j$ and $\mathbf{p}_j$. An explicit formula can be calculated for $P(a; \mathbf{c}_j, \mathbf{p}_j)$ by solving for $N(\mathbf{c}_j, \mathbf{p}_j)$, the number of possible ways the cards can be arranged to satisfy $\mathbf{c}_j, \mathbf{p}_j$. This value satisfies the recursion

$$N(\mathbf{c}, \mathbf{p} + \overline{\delta}_a) = N(\mathbf{c}, \mathbf{p}) - c_a N(\mathbf{c} + \overline{\delta}_a, \mathbf{p})$$

The recursion is solved explicitly in [3], and many other interesting properties of $N(\mathbf{c}, \mathbf{p})$ are explored.

Definition 2.1. The *greedy strategy* is defined as follows. For any configurations $\mathbf{c}_j, \mathbf{p}_j$, guess the jth card according to the function

$$G(\mathbf{c}_j, \mathbf{p}_j) = \text{argmax}_a P(a; \mathbf{c}_j, \mathbf{p}_j).$$

Now, let $E(\mathbf{c}_j, \mathbf{p}_j)$ be the expected number of correct guesses when an optimal strategy is run. Then for $j < n$, E satisfies the recursive formula

$$E(\mathbf{c}_j, \mathbf{p}_j) = \max_a \left(P(a; \mathbf{c}_j, \mathbf{p}_j)(1 + E(\mathbf{c}_j - \overline{\delta}_a, \mathbf{p}_j) + (1 - P(a; \mathbf{c}_u, \mathbf{p}_j)) \cdot E(\mathbf{c}_j, \mathbf{p}_j + \overline{\delta}_a) \right).$$

The optimal strategy $S(\mathbf{c}_j, \mathbf{p}_j)$ satisfies

$$S(\mathbf{c}_j, \mathbf{p}_j) = \text{argmax}_a \left(P(a; \mathbf{c}_j, \mathbf{p}_j) \cdot (1 + E(\mathbf{c}_j - \overline{\delta}_a, \mathbf{p}_j)) + (1 - P(a; \mathbf{c}_j, \mathbf{p}_j) E(\mathbf{c}_j, \mathbf{p}_j + \overline{\delta}_a) \right). \quad (1)$$

This equation is what is used to numerically generate optimal strategies in Section 5.

2.2 Minimizing Entropy

Now let $\mathbf{c}_n$ to denote the exact composition of the remaining $N - n$ cards, after n guesses have been made. Thus $\mathbf{c}_0$ is always assumed known, and after a sequence of guesses $\mathbf{c}_n$ is a random variable, the value of which is not known. The distribution of $\mathbf{c}_n$ is determined by the information gained from the past guesses, and the knowledge the deck was originally uniformly shuffled.

For any card a, let $X_n(a)$ be the indicator that card a is at position n. That is, $X_n(a) = 1$ if card a is at position n, and $X_n(a) = 0$ otherwise. Let I_n denote the information that some sequence of guessing has given us, before guessing card n. More explicitly, $I_n = \{(g_1, X_1(g_1)), \ldots, (g_{n-1}, X_{n-1}(g_{n-1})\}$ is the sequence of guesses g_i and results. Thus, at stage n we assume I_n is just some fixed known value, it is *not* random. We want a strategy that involves minimizing the entropy of the remaining deck $\mathbf{c}_{n+1}$, conditional on the results that the guess on the nth turn gives us, i.e. we want the card a which minimizes

$$H(\mathbf{c}_{n+1} | X_n(a)).$$

In Section 5 we numerically test strategies which choose card a to maximize

$$\mathbb{P}(X_n(a) = 1) - \gamma \cdot H(\mathbf{c}_{n+1} | X_n(a)),$$

for some small constant γ.

2.3 An Example

In this section, we work through an example deck and guessing sequence for which the greedy strategy does not give the optimal guess. This example was given in [1], with the optimal strategy determined by a computer. The optimal strategy we wrote in Section 5 confirms the result.

Consider a deck with initial composition $\mathbf{c}_0 = (3, 3, 3)$, i.e. there are 9 total cards, 3 cards of each of 3 types, say a, b, c. By symmetry, initially any guess will be optimal; so suppose the first guess is card a, and this guess is correct. Thus the exact composition vector is known to be $\mathbf{c}_1 = (2, 3, 3)$. Suppose that the next three guesses are card number b, and all are incorrect (one can check that it is still optimal to guess card b). The fifth guess is again card b, and this guess is correct. Lastly, it is optimal to guess card b again for the sixth guess, and suppose it is correct.

From this information, we can determine the probability distribution on the composition $\mathbf{c}_6$ of the remaining 3 cards. The probabilites are:

- $(0, 1, 2)$, probability $3/10$
- $(1, 1, 1)$, probability $3/5$
- $(2, 1, 0)$, probability $1/10$.

At this point card c has the highest probability of being next, which is $2/5$. However, card b is still the optimal guess, according to the strategy given by Recursion (1), even though the probability it is correct is $1/3$. By considering the possible probability distribution on states of the composition vector $\mathbf{c}_8$ for different guesses for card 7 (for example of this calculation, see Appendix A), we can calculate the conditional entropies for guessing the different cards:

$$H(\mathbf{c}_8|X_7(a)) = 1.777$$
$$H(\mathbf{c}_8|X_7(b)) = 1.079$$
$$H(\mathbf{c}_8|X_7(c)) = 1.643$$

Thus, guessing card b gives the lowest conditional entropy. This example suggests that the optimal strategy may somehow be related to minimizing entropy.

3 No repeated cards

For composition vector $c = (1, \ldots, 1)$, it is known (and intuitively clear) that the greedy strategy is optimal. The strategy can be simply described as "guess a fixed card until it's correct, then pick a new card and guess until it's correct, etc.". You are thus guaranteed to get at least one card correct. Without loss of generality suppose your guessing sequence is $1, 2, 3, \ldots, N$. Then you will get at least k correct guesses if $1, \ldots, k$ are in the correct order in the deck. Since every permutation of these k cards is equally likely, the probability of a fixed order is $\frac{1}{k!}$. Thus if G is the number of correct guesses,

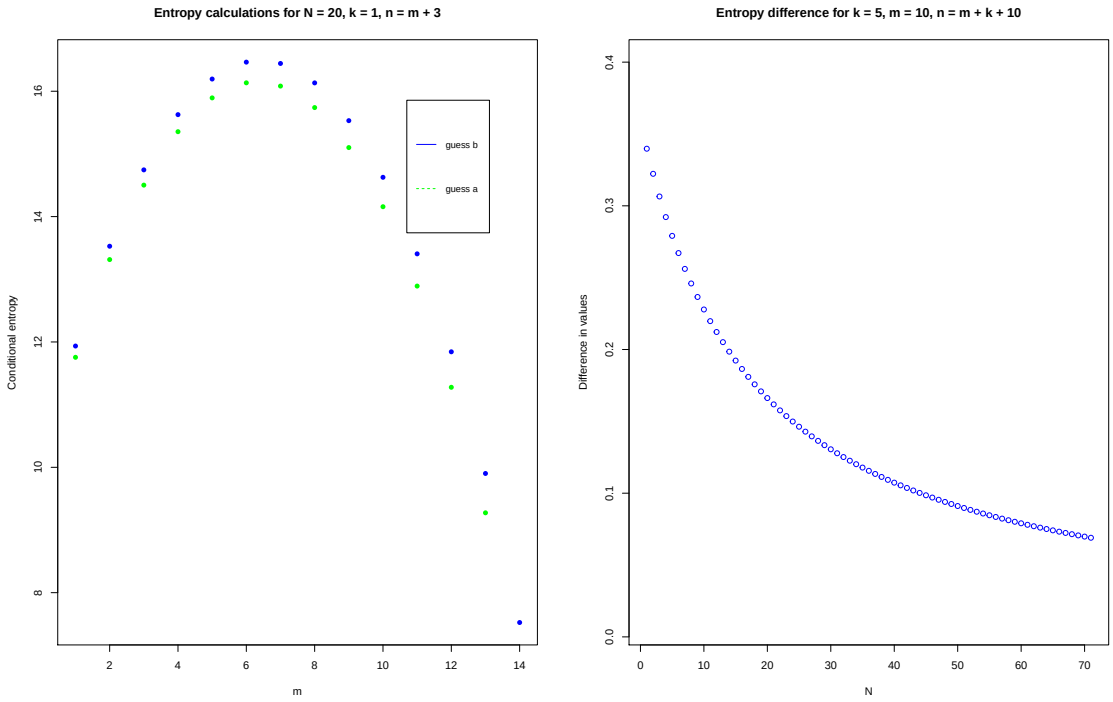
$$\mathbb{E}[G] = \sum_{k=1}^{N} \mathbb{P}(G \geq k) = 1 + \frac{1}{2!} + \cdots + \frac{1}{N!} = e - 1 + O\left(\frac{1}{N!}\right).$$

When $N = 52$, this number is approximately 1.72. The intuition behind this strategy is proven to be optimal in [1].

As a test of the minimize entropy strategy, we can attempt to explicitly calculate and prove that minimizing entropy for a $\mathbf{c} = (1, \ldots, 1)$ deck recovers the optimal strategy. Though this does

not give strong evidence that minimizing entropy is optimal for a general deck, at least it is a step in the right direction. We also include the result and calculation as an example of the types of entropy calculations required for studying this strategy.

3.1 Entropy Calculation



(a) The conditional entropies $H(\mathbf{c}_{n+1}|X_{n+1}(a))$ and $H(\mathbf{c}_{n+1}|X_{n+1}(b))$ in the setting of Theorem 3.1 as m, the number of incorrect guesses of card a, changes.

(b) The difference in conditional entropies $H(\mathbf{c}_{n+1}|X_{n+1}(a)) - H(\mathbf{c}_{n+1}|X_{n+1}(b))$ in the setting of Theorem 3.1 as N, the total number of cards in the deck, increases.

To show that minimizing entropy recovers the optimal strategy, we need to prove that if you've been guessing card a unsuccessfully, guessing card a again at the next step is still optimal. Of course, for the first guess any card guess works. So, assuming that this strategy has been used so far, to use induction we need to prove the general result that if k cards are known so far, and you've guessed card a a fixed number of times, then the entropy conditional on guessing card a again is smaller than the entropy conditional on any other card that has not yet been seen. Note that to guess the final card N, since there are no longer any informational considerations, the strategy is to just resort to greedy and guess the card that has highest probability of being next.

Conjecture 3.1. *Suppose $1 \leq n \leq N-2$ guesses have been made and $k \leq n$ cards have been guessed correctly. Suppose card a is known to be in the deck, and card a was guessed the previous $0 \leq m < (n-k)$ times, all with "no". Let b be any other card that has not yet been guessed. Then,*

$$H(\mathbf{c}_{n+1}|X_{n+1}(a)) \leq H(\mathbf{c}_{n+1}|X_{n+1}(b)).$$

The entropy calculations are messy, but straightforward, and are contained in Appendix A. Once the entropies $H(\mathbf{c}_{n+1}|X_{n+1}(a))$ and $H(\mathbf{c}_{n+1}|X_{n+1}(b))$ are written as functions of N, n, k and

m, it is then difficult to compare the two values to determine when one is larger. We have tested the statement for total number of cards N up to 300 (and all valid values of k, n, m) and it is true; thus, we believe the statement to be true for all values, but state the result as a conjecture since we are not able to prove it analytically.

Figure 1a plots these values when $N = 20, k = 1$ is fixed, m is ranging from 1 to 14 and $n = m+3$. In the figure we can see that guessing card a, the card that has previously been guessed m times, always results in a lower conditional entropy than guessing card b, a card that hasn't yet been guessed. However, we can observe that the magnitude of difference is not large, and indeed, as N (and thus the number of possible states) increases, the difference decreases. This is shown in part Figure 1b, which was computed by fixing n, m, k and increasing N. This suggests that for even moderately sized decks, minimizing entropy alone should not recover the optimal strategy.

4 Bounds on Expected Value Based on Entropy

We can come up with lower and upper bounds for the expected value using the Greedy strategy for the complete feedback case. After searching the literature, we found the following bounds for the upper and lower bounds on the minimal error probability $\pi(X)$ as a function of the entropy [2]

$$\phi^{-1}(H) \geq \pi(X) \geq \Phi^{-1}(H)$$

$$\phi(x) = (M-1)\log(M-1)(1-x) + h((M-1)x - M + 2), \frac{M-2}{M-1} \leq x \leq \frac{M-1}{M}$$

$$\Phi(x) = h(x) + x\log(M-1)$$

$$h(x) = -x\log(x) - (1-x)\log(1-x)$$

Since $\phi(x)$ and $\Phi(x)$ are strictly monotonic functions, they can easily be inverted computationally.

The lower bound on the minimal error probability is a special case of Fano's inequality, and the upper bound is a lemma from [2]. By taking $1 - \pi(X)$, we can find the probability of guessing a card correctly in the game. Since this bound only relates to the probability of guessing correctly for a given state, it can only be applied to the greedy strategy.

Although this can be applied to the greedy strategy in for both incomplete and complete feedback cases, the amount of computation is basically equivalent to calculating the expected value of the greedy strategy explicitly. This is because the probability of each state must be known to calculate the entropy. For the incomplete feedback case, this means the tree of all possible decisions must be traced. We show results for complete feedback where there is no tree since the probability of states does not change with the previous decisions.

To compute the bounds, we do the following:

1. Sweep the two bounding functions $\phi(x)$ and $\Phi(x)$ so we can numerically interpolate to find their inverses.
2. Compute the states at each step of the guessing game and their corresponding probabilities.
3. Compute the entropy at each step j $H(X_j) = \sum_{i=1}^{Total\#states} P(state_i)H(state_i)$
4. Invert the calculated entropy at each step to find $\phi^{-1}(H)$ and $\Phi^{-1}(H)$
5. Compute probability of correct guess at each step j: $P_j(CorrectGuess) = 1 - \pi(X)$

6. Compute expected value of the game: $\mathbb{E}[G] = \sum_j P_j(CorrectGuess)$

We show the results for incomplete feedback with a (3,3,3) deck in Fig. 2. The lower bound (4.70) is close to the actual expected value for the greedy strategy of 4.78.

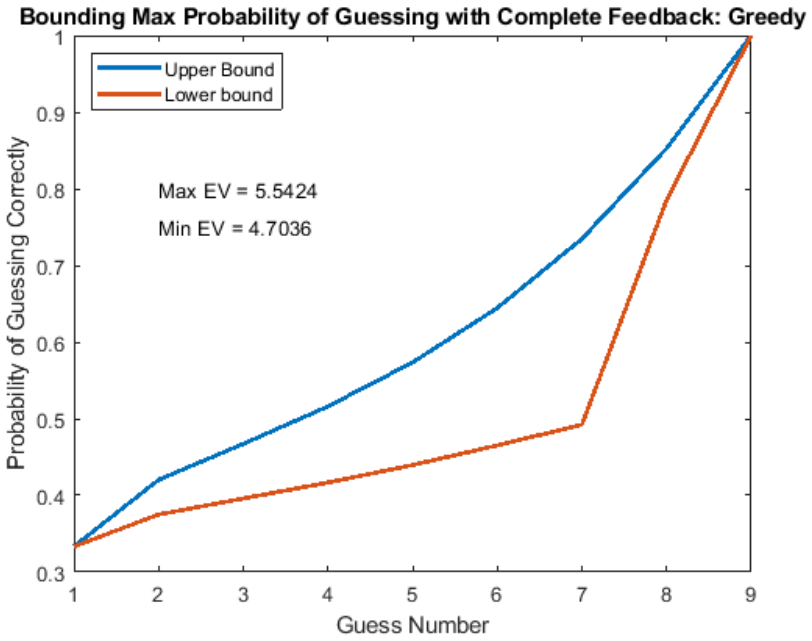


Figure 2: Bounds on the maximum probability of guessing correctly and expected value for complete feedback on (3,3,3) deck

5 Simulation

We wrote a program to simulate the imperfect information card game. This is useful for getting expectation numbers for the greedy and optimal algorithms and also for exploring additional strategies, such as those making better use of information.

The simulator instantiates "agents" that operate by keeping an internal state updated at every card draw and making a decision from that internal state. In the following sections, we describe the state, state updates, and expectation calculations in more detail. We follow that with examples of decision-making based on internal state for specific algorithms.

5.1 State Representation

In this imperfect information game, the brute-force method is to keep track of every possible permutation of cards and the probabilities for each permutation. Initially, they would all have equal probability $\left(\binom{N}{c_1,c_2,\ldots,c_r}\right)^{-1}$. But as the agent makes guesses, certain permutations would get eliminated or become more or less likely.

Unfortunately, this is very computationally costly. A more compressed representation is to record the probabilities of card compositions, getting rid of information about the specific order of cards and only paying attention to how many of each card type are in the remaining portion of the deck.

In code, we represent internal state as a dictionary mapping 'card composition' to 'probability of card composition' with probabilities represented as exact fractions. The following code shows how the internal state is initialized - it starts with the initial compositions, and is updated as described in Section 5.2.

```
self.initial = (3,3,3)
self.states = {self.initial:Fraction(1)}
```

5.2 Imperfect Information Updates

We now describe how to update the internal mixed state given the 'Yes/No' feedback. Updates are performed with Bayes' rule. For example, if we guess card a and receive 'Yes' feedback, then for a given composition state c,

$$P(\text{state} = \text{c} \mid \text{card} = (\text{a, YES})) = \frac{P(\text{card} = (\text{a, YES}) \mid \text{state} = \text{c}) \cdot P(\text{state} = \text{c})}{P(\text{card} = (\text{a, YES}))}$$

All three of the probabilities on the right are easy to calculate. $P(\text{card} = (\text{a, YES}) \mid \text{state} = \text{c})$ is simply the fraction of cards of type a in state c. $P(\text{state} = \text{c})$ is the internal prior probability of state c. $P(\text{card} = (\text{a, YES}))$ is the unconditioned probability of seeing card a given our entire set of internal states (this can be computed by weighing the fraction of card a in each state c by the prior probability of state c).

The update procedure is similar in the case of 'No' feedback, except we need to split the state c into r-1 states, one for each possible choice of alternative card. We weight these choices by their relative frequency in state c (omitting card a, since we condition on its not being the correct choice).

In code, the update looks as follows, where `k` is a state (c), `v` is the prior probability of that state, `guess` is the card we guessed, `correct` is whether we guessed correctly, and `next_state` keeps track of the posterior states.

```
next_states = {}
for k,v in self.states.iteritems():
    if correct and k[guess]:
        bayes = Fraction(k[guess], sum(k)) * v / self.group_prob[guess]
        kl = list(k)
        kl[guess] -= 1
        kl = tuple(kl)
        if kl not in next_states: next_states[kl] = Fraction(0)
        next_states[kl] += bayes
    if not correct and sum(k) - k[guess]:
        bayes0 = (1 - Fraction(k[guess], sum(k))) * v / (1 - self.group_prob[guess])
        for other in range(self.group_num):
            if other == guess or not k[other]: continue
            bayes = bayes0 * Fraction(k[other], sum(k) - k[guess])
            kl = list(k)
            kl[other] -= 1
            kl = tuple(kl)
            if kl not in next_states: next_states[kl] = Fraction(0)
            next_states[kl] += bayes
self.states = next_states
```

5.3 Expectation Calculation

Expectations are calculated recursively by making a guess, spawning two agents who run an expectation calculation on the remaining cards (one conditioned on the guess being correct; the other conditioned on the guess being incorrect), and then weighting the two resulting expectations by the unconditioned probabilities the guesses were correct or not. In code:

```
guess = self.make_guess()

copy0 = deepcopy(self)
if copy0.update_state(guess, False): ev0 = copy0.ev(memo)
else: ev0 = 0

copy1 = deepcopy(self)
if copy1.update_state(guess,  True): ev1 = copy1.ev(memo) + 1
else: ev1 = 0

ev = (1 - self.group_prob[guess]) * ev0 + self.group_prob[guess] * ev1
```

5.4 Greedy Algorithm

The greedy algorithm always selects the card type of highest unconditioned probability. The greedy agent's `make_guess` function thus looks like:

```
def make_guess(self):
    return np.argmax(self.group_prob)
```

Running the expectation calculation, we find:

```
ev = GreedyAgent(initial=(3,3,3)).ev()
print('Expected Correct Guesses: %.15f (%s)'%(ev, str(ev)))
#> Expected Correct Guesses: 4.241071428571429 (475/112)
```

5.5 Optimal Algorithm

For the optimal algorithm, we internally simulate all possible picks and choose the one maximizing expectation. Memoization is used to speed up the simulation/calculation.

```
best_ev = 0
best_pick = 0
for pick in range(self.group_num):
    ev = self.expected_correct(pick, memo, verbose=verbose)
    if ev > best_ev:
        best_ev = ev
        best_pick = pick
```

Running the expectation calculation, we find:

```
ev = OptimalAgent(initial=(3,3,3)).ev()
print('Expected Correct Guesses: %.15f (%s)'%(ev, str(ev)))
#> Expected Correct Guesses: 4.255357142857143 (2383/560)
```

Incredibly, the greedy strategy is not optimal for an initial deck of composition $(3, 3, 3)$! The reason it does better is because occasionally, a hit in probability of getting the next card correct can sometimes provide enough information to be worth it in expectation for the remaining cards. Unfortunately, the optimal algorithm is significantly slower than the greedy algorithm. This leads us to look at a heuristic approach using information.

5.6 Information Heuristics

Like the optimal agent, this agent runs a simulation of what would happen if it picked each card. However, instead of running a full recursion tree, we only look one level down to see the resultant state entropy. We then calculate a simple modified heuristic of expected value minus γ times future entropy and maximize over all card picks.

```
best_eiv  = 0
best_pick = 0
for pick in range(self.group_num):
    copy0 = deepcopy(self)
    copy0.update_state(pick, False)
    ent0 = copy0.get_state_entropy()
    copy1 = deepcopy(self)
    copy1.update_state(pick, True)
    ent1 = copy1.get_state_entropy()
    entropy = (1 - self.group_prob[pick]) * ent0 + self.group_prob[pick] * ent1
    eiv = self.group_prob[pick] - self.gamma * entropy
    if eiv > best_eiv:
        best_eiv = eiv
        best_pick = pick
```

Running the expectation calculation, we find:

```
ev = InfoAgent(gamma=0.15, initial=(3,3,3)).ev()
print('Expected Correct Guesses: %.15f (%s)'%(ev, str(ev)))
#> Expected Correct Guesses: 4.255357142857143 (2383/560)
```

For this particular initial deck and setting for γ, the information heuristic agent actually matches the optimal algorithm! In Figure 3, we show the effect of increasing γ on expected correct guesses. When γ is small, information is not prioritized enough, leading to greedy strategies. When γ is large, information is prioritized too much, leading to too little exploitation of current information.

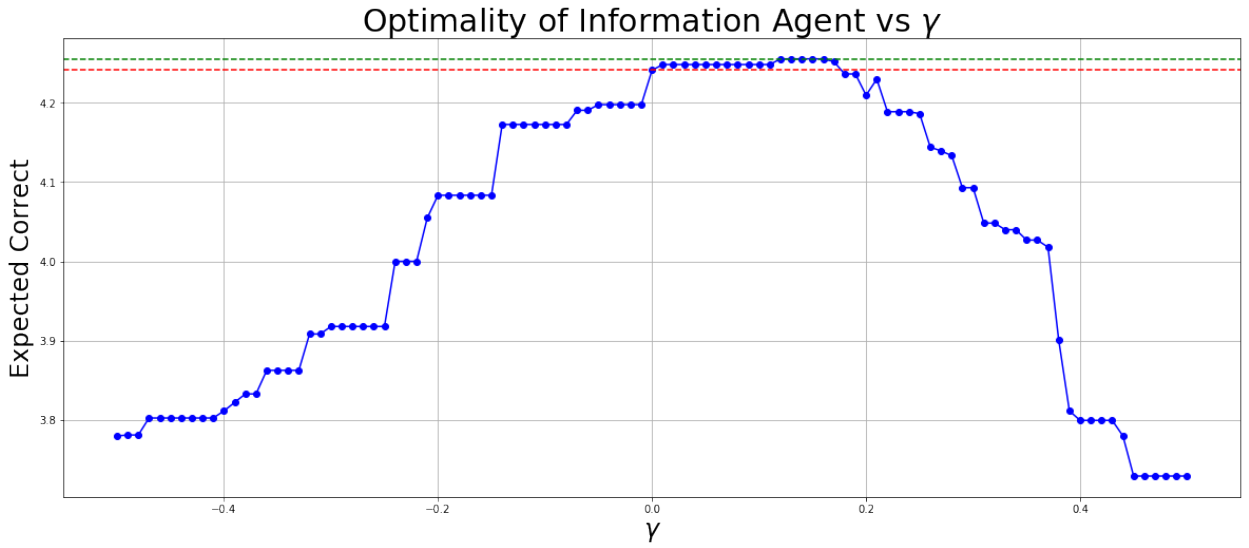


Figure 3: Expectation of info theory agent (blue) versus γ. The greedy agent (red, dashed) and optimal agent (green, dashed) are shown for comparison. The info theory agent achieves optimality for $\gamma \approx 0.15$ in this case.

6 Simple Playable Strategies

The strategies discussed such as the optimal strategy or the greedy strategy requires the player to keep track of the possible states and their probabilities, which is difficult to implement when actually playing the game (unless you play with a computer). In this section, we will discuss some simple strategies that could be implemented by a person while playing the game and their simulated performance.

One idea is to try to keep track of the expected number of cards for each type left in the deck. This strategy requires the player to only keep track of three numbers, and modify the numbers according to the yes/no feedback. The player can update it is as follows:

1. If the guess is correct, subtract one from the type of card guessed
2. If the guess is incorrect, then for each of the types not guessed we denote the number of cards of the type by C_i. The update is $C_i = C_i - C_i/(\sum_i C_i)$

This strategy is similar to the greedy strategy, but actually some information is lost. The number of expected cards can become negative, at which point calculating the probabilities do not make sense. If the value is set to zero, then some information is lost.

This is reflected in the simulated results. For the above strategy on a (3,3,3) deck, the expected value is 4.2387, which is lower than the greedy expected value of 4.241. Interestingly, to make the computation simpler for the player, we can replace the subtraction in step 2 by a constant: $C_i = C_i - 0.55$ which has an expected value of 4.2482. This is higher than greedy for this particular deck.

7 Outreach

For the outreach event at Lucille M. Nixon Elementary School, we wanted to teach kids about probability and how to think about strategies for guessing games. Since our project was about a card guessing game, we had them play the incomplete feedback game, and afterwards we explained how to implement an effective strategy to maximize the number of correctly guessed cards.

The kids played the card game with 3 Jacks, 3 Queens and 3 Kings with incomplete feedback. In the beginning we had the computer simultaneously play along with the kids using the optimal strategy. We created a modified version of the simulation that outputted the possible states along with their probabilities in a bar graph format for easy viewing. After a few kids, we changed the way we played the game with the kids to use the simulation to explain how the optimal strategy works.

We noticed that the kids and their parents easily forgot what cards they had previously guessed and what the results were (and then proceeded to guess randomly). To help them with this we decided to place each guessed card on the table: face up if they guessed correctly and face down if they guessed incorrectly. This made the game more visual. In addition, one of us would write down the guess they made so that they can see their incorrect guesses as well.

Second, we also realized how short we could keep the kids' attention for. To address this, after they had finished guessing the deck, we would flip all the cards over and show how the computer simulation would play on the same deck. This gives us a chance to point out how the states evolve when a correct guess is made, and how the states split when an incorrect guess is made. Finally the bar graph was an easy visual way to show this how this information would be used to determine the next guess in a tractable way for the kids. An example case of the output we showed to the kids when the computer played the game is shown in Fig. 4.

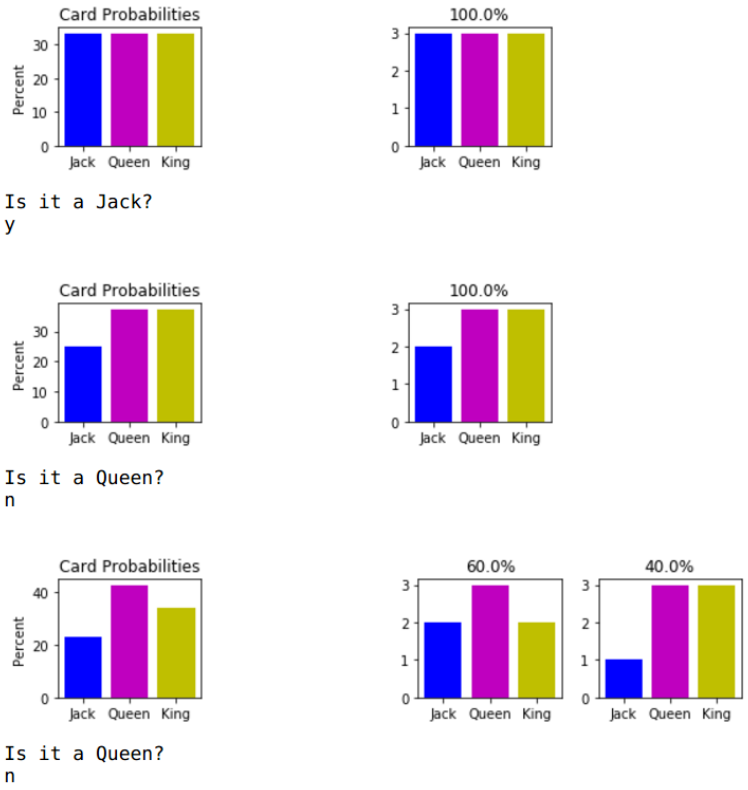


Figure 4: Interactive CPU opponent for outreach, showing internal state for teaching purposes.

For example, in Fig. 4, we would point out that the CPU correctly guessed Jack, so now it knew there was a 100% chance of being in the (2,3,3) state. However, when it incorrectly guessed Queen, two possible states emerged: one where Jack was the correct card (1,3,3) and one where King was the correct card (2,3,2). We would also point out how because there were fewer Jacks to start with, the (1,3,3) state was less likely than the (2,3,2) state.

When the CPU guessed the first card incorrectly, we would jump immediately into this "No" case. When the CPU guessed a sequence of card correctly, we would speed ahead until the first "No." After showing an example of state splitting at a "No," we would speed through the rest of the cards to get the final result, which we compared to the student's scores.

We recorded the results from the outreach, and we found that on average, the computer had an advantage compared to the kids. The results are summarized in Table 1. We only took one photo of the outreach event, which was the view from our booth (Fig. 5).

Player Number	Guesses Correct (Human)	Guess Correct (CPU)
1	5	2*
2	3	3
3	5	4
4	3	4
5	0	5
6	2	3
7	6	5
8	3	6
9	2	3
Average	3.222	3.889

Table 1: Results of outreach guessing game. On average humans did 17% worse than the CPU. We input the data incorrectly into the CPU for the first game, which resulted in an erroneously low CPU score. Omitting this data point, humans would have performed 27% worse.

Figure 5: Photograph of outreach event at Lucille M. Nixon Elementary School.

8 Conclusion

This simple card game is a metaphor for life. Everyone goes through life attempting to maintain an accurate model of the world in order to make decisions and take actions. However, to maintain accuracy, we must apply Bayesian updates to our prior models based on imperfect information. And in making our choices, we can prioritize immediate rewards or we can prioritize information and learning, for potentially more accurate internal models and potentially higher future expected rewards.

Some people may be frozen by the inherent fuzziness of life, but this game teaches us that sometimes we can push luck in our favor, making risk-taking valuable. We saw this in our outreach event, where the students who did well tended to be the ones willing to take smart risks based on whatever information they were able to gather from their previous guesses. While life may be stochastic and arbitrary, you can still improve your odds by using information to the best of your

advantage.

A Entropy with No Repeated Cards

Suppose we are in the setting of Theorem 3.1. Note that the probability of observing the information is

$$\mathbb{P}(I_n) = \frac{(N-m-k)\cdot(N-k-1)!}{N!} = \frac{N-m-k}{N\cdot(N-1)\dots(N-k)}.$$

Entropy after guessing a First, we calculate the conditional entropy if you guess card a again at the $n+1$th step. The probability that card a is correct is

$$\mathbb{P}(X_{n+1}(a)=1|I_n) = \frac{\mathbb{P}(X_{n+1}(a)=1\,\&\,I_n)}{\mathbb{P}(I_n)} = \frac{1}{N-m-k}.$$

If card a is correct, there are $\binom{N-k-1}{n-k}$ possibilities for the remaining composition $\mathbf{c}_{n+1}$: we know $n-k$ cards have been passed, and there are $N-k-1$ possibilities for which cards they were. Each possibility is equally likely, so

$$H(\mathbf{c}_{n+1}|X_{n+1}(a)=1) = \log\left(\binom{N-k-1}{n-k}\right). \tag{2}$$

If card a is guessed and it is incorrect, there are two different types of potential configurations $\mathbf{c}_{n+1}$:

- Card a is still in the deck. There are $\binom{N-k-1}{n-k+1}$ possibilities for $\mathbf{c}_{n+1}$ that give this, and each is equally likely. The probability that a is still in the deck is given by

$$\mathbb{P}(\mathbf{c}_{n+1}(a)=1|I_n, X_{n+1}(a)=0) = \frac{N-(n+1)}{N-(m+k+1)}.$$

- Card a is not remaining the deck. This probability is $1-\frac{N-(n+1)}{N-(m+k+1)}$. There are $\binom{N-k-1}{n-k}$ configurations in this setting, and each is equally likely.

Thus,

$$\begin{aligned} H(\mathbf{c}_{n+1}|X_{n+1}(a)=0) = &-\frac{N-(n+1)}{N-(m+k+1)}\log\left(\frac{N-(n+1)}{\binom{N-k-1}{n-k+1}\cdot(N-(m+k+1))}\right) \\ &-\frac{n-k}{N-(m+k+1)}\log\left(\frac{n-k}{\binom{N-k-1}{n-k}\cdot(N-(m+k+1))}\right) \end{aligned} \tag{3}$$

Thus, the complete conditional entropy can be calculated using

$$H(\mathbf{c}_{n+1}|X_n(a)) = \frac{1}{N-m-k}H(\mathbf{c}_{n+1}|X_{n+1}(a)=1) + \frac{N-m-k-1}{N-m-k}H(\mathbf{c}_{n+1}|X_{n+1}(a)=0. \tag{4}$$

Entropy after guessing b Now suppose that for card $n+1$ we guess any other card b that has not been previously guessed. First, conditional on our information I_m, the probability that b is correct is

$$\mathbb{P}(X_{n+1}(b)=1|I_m) = \frac{N-(m+k+1)}{(N-(m+k))(N-(k+1))}.$$

Consider first the case that $X_{n+1}(b)=1$. There are two main possibilities for the remaining composition:

- Card a is still in the deck. The probability of this is

$$\mathbb{P}(\mathbf{c}_{n+1}(a) = 1 | I_n, X_{n+1}(b) = 1) = \frac{N - (n+1)}{N - (m+k+1)}.$$

  Each way that this happens is equally likely, and there are $\binom{N-k-2}{n-k}$ ways.

- Card a is not in the deck. There are $\binom{N-k-2}{n-k-1}$ such ways, each equally likely.

Thus, the conditional entropy on this event is

$$\begin{aligned} H(\mathbf{c}_{n+1} | X_{n+1}(b) = 1) = &- \frac{N-(n+1)}{N-(m+k+1)} \log \left(\frac{1}{\binom{N-k-2}{n-k}} \cdot \frac{N-(n+1)}{N-(m+k+1)} \right) \\ &- \frac{n-m-k}{N-m-k-1} \log \left(\frac{1}{\binom{N-k-2}{n-k-1}} \cdot \frac{n-m-k}{N-m-k-1} \right) \end{aligned} \tag{5}$$

Now consider the case $X_{n+1}(b) = 0$. This results in the most complicated probability distribution for $\mathbf{c}_{n+1}$. The complication arises in the case that the true card at position $n+1$ was a. It is helpful to compute:

$$\begin{aligned} \mathbb{P}(I_n, X_{n+1}(b) = 0) &= \frac{(N-(k+1))! + (N-(k+m+1)) \cdot (N-(k+2)) \cdot (N-(k+2))!}{N!} \\ &= [(N-(k+1)) + (N-(k+m+1)) \cdot (N-(k+2))] \cdot \frac{(N-(k+2))!}{N!} \end{aligned}$$

The first term in the sum is the case that card $n+1$ is a. For ease of writing, say $A = [(N-(k+1)) + (N-(k+m+1)) \cdot (N-(k+2))]$. We consider the following cases:

1. Position $n+1$ was card a and card b is still in the deck. There are $\binom{N-(k+2)}{n-k}$ ways. The probability is

$$\mathbb{P}(\mathbf{c}_{n+1}(b) = 1, X_{n+1}(a) = 1 | I_n, X_{n+1}(b) = 0) = \frac{N-(n+1)}{A}.$$

2. Position $n+1$ was card a and card b is not still in the deck. There are $\binom{N-(k+2)}{n-k-1}$ ways, with probability

$$\mathbb{P}(\mathbf{c}_{n+1}(b) = 0, X_{n+1}(a) = 1 | I_n, X_{n+1}(b) = 0) = \frac{n-k}{A}.$$

3. Position $n+1$ was not card a, and card a and card b are both still in the deck. There are $\binom{N-(k+2)}{n-k+1}$ ways. Probability is

$$\mathbb{P}(\mathbf{c}_{n+1}(a) = 1, \mathbf{c}_{n+1}(b) = 1 | I_n, X_{n+1}(b) = 0) = \frac{[N-(n+1)] \cdot [N-(n+2)]}{A}.$$

4. Position $n+1$ was not card a, and card a is still in the deck, but card b is not. There are $\binom{N-(k+2)}{n-k}$ ways. The probability of this is

$$\mathbb{P}(\mathbf{c}_{n+1}(a) = 1, \mathbf{c}_{n+1}(b) = 0 | I_n, X_{n+1}(b) = 0) = \frac{[N-(n+1)] \cdot [n-k]}{[N-(k+1)] \cdot [N-(k+m)]}.$$

5. Position $n+1$ was not card a, and card a is not in the deck, but b still is. There are $\binom{N-(k+2)}{n-k}$ ways. Probability is

$$\mathbb{P}(\mathbf{c}_{n+1}(a)=0,\mathbf{c}_{n+1}(b)=1,X_{n+1}(a)=0|I_n,X_{n+1}(b)=0)=\frac{[n-(m+k)]\cdot[N-(n+1)]}{A}$$

6. Position $n+1$ was not card a, and card a and card b are both not in the deck. There are $\binom{N-(k+2)}{n-k-1}$ ways. The probability of this is

$$\mathbb{P}(\mathbf{c}_{n+1}(a)=0,\mathbf{c}_{n+1}(b)=0,X_{n+1}(a)=0|I_n,X_{n+1}(b)=0)=\frac{[n-(m+k)]\cdot[n-(k+1)]}{A}.$$

Thus, we have all the ingredients necessary to calculate $H(\mathbf{c}_{n+1}|X_n(b))$.

References

[1] Diaconis, Persi, and Ronald Graham. "The analysis of sequential experiments with feedback to subjects." The Annals of Statistics 9.1 (1981): 3-23.

[2] Feder, Meir, and Neri Merhav. "Relations between entropy and error probability." IEEE Transactions on Information Theory 40.1 (1994): 259-266.

[3] F.R.K Chung, P Diaconis, R.L Graham, and C.L Mallows. "On the permanents of complements of the direct sum of identity matrices." Advances in Applied Mathematics, 2(2) (1981):121 137.