The AAAI-17 Workshop on
Computer Poker and Imperfect Information Games
WS-17-06

Learning Human-Understandable Strategies

Sam Ganzfried and Farzana Yusuf
School of Computing and Information Sciences
Florida International University
{sganzfri@cis.fiu.edu, fyusu003@fiu.edu}

Abstract

Algorithms for equilibrium computation generally make no attempt to ensure that the computed strategies are understandable by humans. For instance the strategies for the strongest poker agents are represented as massive binary files. In many situations, we would like to compute strategies that can actually be implemented by humans, who may have computational limitations and may only be able to remember a small number of features or components of the strategies that have been computed. We study poker games where private information distributions can be arbitrary. We create a large training set of game instances and solutions, by randomly selecting the private information probabilities, and present algorithms that learn from the training instances in order to perform well in games with unseen information distributions. One approach first clusters the training points into a small number of clusters and then creates a small decision tree based on the cluster centers. This approach produces low test error and could be easily implemented by humans since it only requires memorizing a small number of "if-then" rules.

1 Introduction

Large-scale computation of strong game-theoretic strategies is important in many domains. For example, there has been significant recent study on solving game-theoretic problems in national security from which real deployed systems have been built, such as a randomized security check system for airports (Paruchuri et al. 2008). Typically large-scale equilibrium-finding algorithms output massive strategy files (which are often encoded in binary), which are stored in a table and looked up by a computer during gameplay. For example, the recently computed optimal strategy for two-player limit Texas hold 'em requires 262 TB of storage (using 4-byte floating-point numbers) (Bowling et al. 2015). While such approaches can lead to very strong computer agents, it is difficult to see how a human could implement these strategies. For cases where humans will be making real-time strategic decisions we would like to compute strategies that are easily interpretable and understandable.

Suppose a human plans to play the following two-player no-limit poker game. Player 1 and player 2 both ante $0.50 and are dealt a card from a 10-card deck and each have a stack of $3 after posting the ante. Player 1 can bet any multiple of 0.1 from 0 to 3 (he has 31 possible actions for each hand). Player 2 can then call or fold. If player 2 folds, then player 1 wins the $1 from the antes. Otherwise the player with the better card wins the amount bet plus the antes. For example, if player 1 has a 4, player 2 has a 9, player 1 bets 0.4 and player 2 calls, then player 2 wins 0.4 plus the antes.

If both players are dealt cards uniformly at random (Figure 1), then the following is a Nash equilibrium strategy for player 1:

- Card 1: Bet 0.1 prob 0.091, 0.6 prob 0.266, 1.8 prob 0.643
- Card 2: Bet 0 prob 0.660, 0.3 prob 0.231, 0.6 prob 0.109
- Card 3-6: Bet 0 prob 1
- Card 7: Bet 0.1 prob 1
- Card 8: Bet 0.3 prob 1
- Card 9: Bet 0.6 prob 1
- Card 10: Bet 1.8 prob 1

This can be computed quickly using, e.g., a linear programming formulation (Koller and Megiddo 1992).

However, suppose the cards are dealt according a different distribution: player 1 is either dealt a very strong hand (10) or a very weak hand (1) with probability 0.5 while player 2 is always dealt a medium-strength hand (Figure 2). Then the equilibrium strategy for player 1 is:

- Card 1: Bet 0 prob 0.25, 3 prob 0.75
- Card 10: Bet 3 prob 1

If player 1 is always dealt a medium-strength hand (5) while player 2 is dealt a very strong or very weak hand with probability 0.5 (Figure 3), then the equilibrium strategy is:

- Card 5: Bet 0 prob 1

What if player 1 is dealt a 1 with probability 0.09, 2 with probability 0.19, 3 with probability 0.14, etc.? For each game instance induced by a probability distribution over the private information, we could solve it quickly if we had access to an LP solver. But what if a human is to play the game without knowing the distribution in advance and without aid of a computer? He would need to construct a strong game plan in advance that is capable of playing well for a variety of distributions with minimal real-time computation. A natural approach would be to solve and memorize solutions for

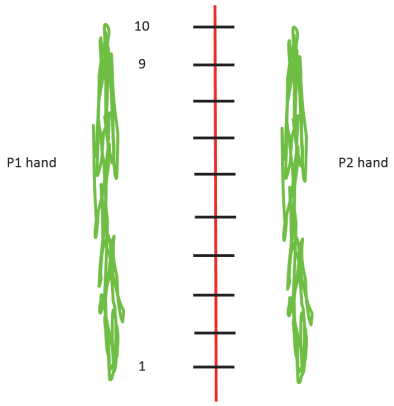

Figure 1: Both players are dealt private information uniformly at random over all hands.

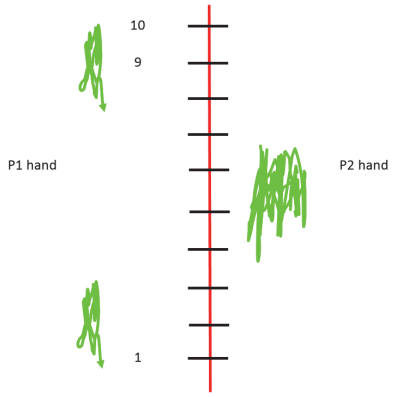

Figure 2: Player 1 is dealt very strong or weak hand and player 2 is always dealt mediocre hand.

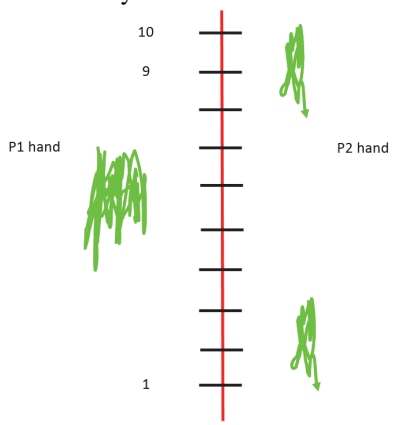

Figure 3: Player 1 is always dealt mediocre hand and player 2 is dealt very strong or weak hand.

several games in advance, then quickly determine which of these games is closest to the one actually encountered. This is akin to the k-nearest neighbors (k-nn) algorithm from machine learning. A second would be to construct understandable rules (e.g., if .. else ..) from a database of solutions that can be applied to a new game. This is akin to the decision tree and decision list approaches.[1] Thus, we are proposing to apply approaches from machine learning in order to improve human ability to implement Nash equilibrium strategies. Typically algorithms from machine learning have been applied to game-theoretic agents only in the context of learning to exploit mistakes of suboptimal opponents (aka opponent exploitation). By and large the approaches for computing Nash equilibrium and opponent exploitation have been radically different. We provide a new perspective here by integrating learning into the equilibrium-finding paradigm.

We present a novel learning formulation of this problem. In order to apply algorithms we develop novel distance functions (both between pairs of input points and between pairs of output points) which are more natural for our setting than standard distance metrics. To evaluate our approaches we compute a large database of game solutions for random private information distributions. We are able to efficiently apply k-nn to the dataset using our custom distance functions. Experiments show that we are able to obtain low testing error even when training on a relatively small fraction of the data, which suggests that it is possible for humans to learn strong strategies by memorizing solutions to a carefully selected small set of presolved games. We also investigate decision trees to compute human understandable rules, and present the most prominent rule, which has depth up to level 6. Finally, we explore clustering the training instances into 5, 10, and 15 clusters, and then computing a decision tree to implement a strategy based on the cluster centers. This approach produces relatively low test error and only involves memorizing a very small number of "if-then" rules from the optimal decision tree, thus making it easy for a human to implement this strategy in an arbitrary game instance.

While prior approaches for learning in games of imperfect information (and poker specifically) typically utilize many poker-specific features (e.g., number of possible draws to a flush), we prefer to develop approaches that are more robust and do not require knowing expert domain features (since they are likely not relevant for other domains and, in the case of poker, may not be relevant even for other seemingly similar variants). The features we use are the cumulative distribution function (cdf) values of the private information states of the players, which are based purely on the rules of the game. (We also compare performance of using several other data representations, e.g., using pdf values, and separating out the data for each hand to create 10 data points per game instance instead of 1). Thus, the approach is general and not reliant on expert poker knowledge.

2 Qualitative models and endgame solving

There has been some prior study of human understandable strategies in imperfect-information games, and in poker specifically. In "Mathematics of Poker," Ankenman and Chen compute analytical solutions of several simplified poker variants (which typically assume continuous uniform private information distributions) by first assuming a given

[1]The problem of constructing human-interpretable rules has also been studied in machine learning, e.g., (Bertsimas, Chang, and Rudin 2011).

human-understandable qualitative structure on the equilibrium strategies, and then computing equilibrium strategies given this presumed structure, typically by solving a series of indifference equations (Ankenman and Chen 2006). While the computed strategies are generally interpretable by humans, the qualitative equilibrium models were typically constructed from a combination of trial and error and expert intuition, and not constructed algorithmically. More recent work has shown that leveraging such qualitative models can lead to new equilibrium-finding algorithms that outperform existing approaches (Ganzfried and Sandholm 2010). That work proposed three different qualitative models for the final round (river) endgame of two-player limit Texas hold 'em (Figures 4– 6), and showed empirically that equilibrium strategies for the endgame conformed to one of the models for all input distributions of private information (and that all three were needed). Again here the models were constructed by manual trial and error, not learned algorithmically.

We note that while the problem we are considering in this paper is a "toy game," it captures important aspects of real poker games and we expect our approaches to have application to larger and more realistic variants, in addition to domains besides poker due to the generality (our new distance functions and approaches could be of independent interest). In the recent Brains vs. Artificial Intelligence two-player no-limit Texas hold 'em competition, the agent Claudico computed the strategy for the final betting round in real time, and the best human player in the world for that variant (Doug Polk) commented that the "endgame solver" was the strongest component of the agent (Ganzfried 2015). The endgame solving algorithm assumed that both agents had private information distributions induced by the strategies for the prior rounds using Bayes' rule, assuming both agents had been following the agent's strategy for the prior rounds (Ganzfried and Sandholm 2015). The game we study here is very similar to no-limit Texas hold 'em endgames, except that we are assuming a ten-card deck, specific stack sizes and betting increment, and that raises are not allowed. We expect our analysis to extend in all of these dimensions and that our approaches will have implications for no-limit Texas hold 'em strategy (particularly for final round endgames and potentially for earlier rounds too). No-limit Texas hold 'em is the most popular poker variant for humans, and is a widely recognized AI challenge problem. The game tree has approximately 10^{165} states for the variant played in the AAAI Annual Computer Poker Competition (Johanson 2013). There has been significant interest in endgame solving in particular in the last several years, and several new advances have been developed (Burch, Johanson, and Bowling 2014; Moravcik et al. 2016).

3 Learning formulation

We now describe how we formulate the problem of computing a solution to a new game instance from a database of solutions to previously solved game instances as a learning problem. The inputs to the learning problem will be the 20 values of the private information cumulative distribution function (cdf). First are the ten values for player 1 (the probability he is dealt ≤ 1, probability he is dealt ≤ 2, etc.), followed by the ten cdf values for player 2. For example for the uniform case the input would be

$$X = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1,$$
$$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1),$$

for the situation where player 1 is dealt a 10 or 1 with probability 0.5 and player 2 is always dealt a 5 it is

$$X = (0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1,$$
$$0, 0, 0, 0, 1, 1, 1, 1, 1, 1),$$

and for the situation where player 1 is always dealt a 5 and player 2 is dealt a 10 or 1 with probability 0.5 it is

$$X = (0, 0, 0, 0, 1, 1, 1, 1, 1, 1,$$

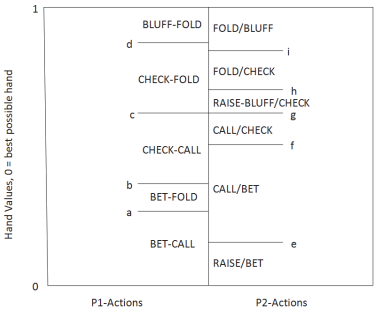

Figure 4: First qualitative model for two-player limit Texas hold 'em river endgame play.

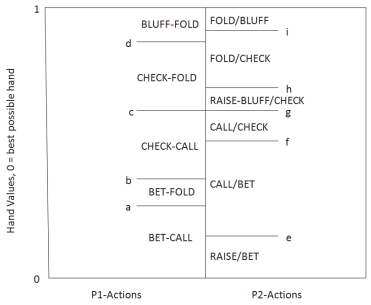

Figure 5: Second qualitative model for two-player limit Texas hold 'em river endgame.

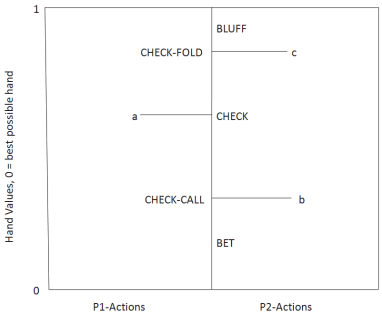

Figure 6: Third qualitative model for two-player limit Texas hold 'em river endgame.

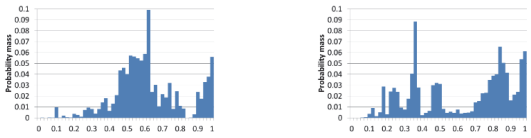

Figure 7: Earth-mover's distance has proven a successful metric for distance between probability distributions.

$$0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1).$$

The output will be a vector of the 310 Nash equilibrium strategy probabilities of betting each size with each hand. First for betting 0, 0.1, 0.2, ..., 3 with 1, then with 2, etc. (recall that there are 31 sizes for each of ten hands). For example for the uniform case the output would be

$$y = (0, 0.091, 0, 0, 0, 0, 0.266, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0.643, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots).$$

We could have created ten different data points for each game corresponding to the strategy for each hand, as opposed to predicting the full strategy for all hands; however we expect that predicting complete strategies is better than just predicting strategies for individual hands because the individual predicted hand strategies may not balance appropriately and could be highly exploitable as a result. We will explore this design choice in the experiments in Section 4.

To perform learning on this formulation, we need to select a distance function to use between a pair of inputs as well as a distance (i.e., cost) between each pair of outputs. Standard metrics of Euclidean or Manhattan distance are not very appropriate for probability distributions. A more natural and successful distance metric for this setting is earth mover's distance (EMD). While early approaches for computing groupings of hands used L2 (Gilpin, Sandholm, and Sørensen 2007), EMD has been shown to significantly outperform other approaches, and the strongest current approaches for game abstraction use EMD (Johanson et al. 2013). Informally, EMD is the "minimum cost of turning one pile into the other, where the cost is assumed to be amount of dirt moved times the distance by which it is moved," and there exists a linear-time algorithm for computing it for one-dimensional histograms (Figure 7).

We define a new distance metric for our setting that generalizes EMD to multiple distributions. Suppose we want to compute the distance between training input vector X and testing input vector $\hat{X}$. Each vector contains 20 probabilities, 10 corresponding to player 1's distribution and 10 to player 2's. Our distance function will compute the EMD separately for each player, then return the average (Algorithm 1). We note that before the aggregation we normalize the EMD values by the maximum possible value (the distance between a point mass on the left-most and right-most columns) to ensure that the maximum of each is 1. We also create a new distance (i.e., cost) function between predicted output vector $\hat{Y}$ and the actual output vector from the training data Y (the output vectors have length 310, corresponding to 31 bet sizes for 10 hands). It computes EMD separately for the strategy vectors of size 31 for each hand which are then normalized and averaged (Algorithm 2). After specifying the form of the inputs and outputs and a distance metric between each pair of inputs and outputs, we have formulated the problem as a machine learning problem.

Algorithm 1 Distance between input vectors X, $\hat{X}$

Inputs: cdf vectors X, $\hat{X}$, number of players n, deck size d

```
X' ← cdf-to-pdf(X); X̂' ← cdf-to-pdf(X̂)
resultTotal ← 0
for i = 0 to n − 1 do
    start ← i × d, end ← start + d
    result ← 0, δ ← 0
    for j = start to end-1 do
        δ ← δ + X'[j] − X̂'[j]
        result ← result + |δ|
    result ← result / (d-1)
    resultTotal ← resultTotal + result
resultTotal ← resultTotal / n
return resultTotal
```

Algorithm 2 Distance between output vectors Y, $\hat{Y}$

Inputs: Strategy vectors $Y, \hat{Y}$, deck size d, number of bet sizes b

```
resultTotal ← 0
for i = 0 to d − 1 do
    start ← i × b; end ← start + b
    result ← 0, δ ← 0
    for j = start to end-1 do
        δ ← δ + Y[j] − Ŷ[j]
        result ← result + |δ|
    result ← result / (b-1)
    resultTotal ← resultTotal + result
resultTotal ← resultTotal / d
return resultTotal
```

4 Experiments

We constructed a database of 100,000 game instances by generating random hand distributions and then computing a Nash equilibrium using the linear program formulation with Gurobi's solver (Gurobi Optimization, Inc. 2014). The naïve approach for constructing the distributions (of assigning uniform distributions for the players independently) is incorrect because it does not account for the fact that if one player is dealt a card then the other player cannot also be dealt that card (as is the case in real poker). We instead used a new procedure described in Algorithm 4. We first generate the two distributions independently as in the naïve approach, using the procedure described in Algorithm 3. Algorithm 4 then multiplies these individual probabilities together only for situations where players are dealt different cards to compute a joint distribution over the private information (these values are then normalized). The procedure could be more generally applicable beyond this setting. We then create the cdf values from the joint private information distribution to be used as the inputs to the learning problem.

We experimented with several different data representations. The first was described above. The second uses the

Algorithm 3 Generate point uniformly at random from n-dimensional simplex

Inputs: dimension n

$s = 0$
for $i = 0$ to $n - 1$ **do**
  $a[i] \leftarrow$ randomDouble(0,1)
  $a[i] \leftarrow -1 \times \log(a[i])$
  $s \leftarrow s + a[i]$
for $i = 0$ to $n - 1$ **do**
  $a[i] \leftarrow a[i]/s$
return a

Algorithm 4 Generate private information distribution

Inputs: dimension n, independent distributions x_1, x_2

$s = 0$
for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $n - 1$ **do**
    if i != j **then**
      next $\leftarrow x_1[i] \times x_2[j]$
      $x^*[i][j] \leftarrow$ next
      $s \leftarrow s +$ next
for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $n - 1$ **do**
    $x^*[i][j] \leftarrow x^*[i][j]/s$
return x^*

pdf values as the 20 features instead of the cdfs. The third separates each datapoint into 10 different points, one for each hand of player 1. Here the first 20 inputs are the cdfs as before, followed by a card number (1–10), which can be viewed as an additional 21st input, followed by the 31 strategy probabilities for that card. The fourth uses this approach with the pdf features. The 5th and 6th approaches are similar, but for the 21st input they list the cdf value of the card, not the card itself.

4.1 Nearest neighbors

We first experimented with k-nearest neighbors (k-nn). We selected this algorithm because a natural approach for humans would be to study solutions to a number of situations in advance and then map the situation encountered in real time to one previously studied. In particular we would expect that humans would naturally implement the strategies from the single closest solution that was studied, and therefore we focused on $k = 1$. For our experiments, we used cross validation, varying the division percentages between the training and testing set to examine the effect of training data size on test error. Due to the complexity of k-nn (it must compute distances between each test input and each training input), we selected only a subset of the database for these experiments (1000 games). We used our custom distance function for the inputs and outputs. The results are in Figures 8 and 9. The pdf representation produced slightly lower errors, which was expected because it encodes more information than the cdf (the pdfs encode all of the information for the hand distributions from the original game, while the cdfs do not correspond to unique hand distributions—that is, it is possible for several games with different hand probabilities to produce the same cdf values but not the same pdfs). We note that for both approaches, even using a small fraction of the training data already produced relatively low test error, which suggests that it may be possible to generate a small set of carefully chosen instances for humans to study and perform well in a variety of unknown instances. We also observed that using the original 310 representations outperformed the 31 ones. Figures 10 and 11 show the results for the latter representation. The large spikes in error as a function of training set size indicate that we should likely experiment on more games.

4.2 Decision tree

The next approach we considered was to learn a decision tree. For this approach we used 20,000 of the games from the database, using the standard division of 80% the data for training and 20% for testing. We used Python's built in decision tree regressor function from sklearn.tree from the scikit-learn library. This library uses mean squared error as a default, but we were able to integrate our new distance metrics (we can do this also for the built-in libraries for other common machine learning algorithms). We report the errors using our new generalized EMD distance function, as well as the mean-squared error score, which is the "coefficient of determination R^2 of the prediction."

Experimenting on all 6 representations, the 5th (using cdfs separately for each card plus a separate card cdf input) produced lowest error (Table 1), though all approaches produced extremely low errors. We expected using the cdfs to perform better for the decision trees since they seem like natural values to branch on, despite the fact that for k-nn pdf performed better and using 310 outputs outperformed just using 31. Using Python's DecisionTreeRegressor function with a maximum depth of 12 produced an EMD error of 6.68×10^{-8} for the best approach on 100,000 games.

Representation	Error (EMD)	Score (MSE)
pdf-310	0.001123239	0.083686509
cdf-310	0.000818163	0.110532576
pdf-card-31	2.79×10^{-6}	0.164877403
cdf-card-31	6.68×10^{-8}	0.197207893
pdf-card-cdf-31	0.000145299	-0.15212355
cdf-card-cdf-31	3.29×10^{-6}	0.165946891

Table 1: Earth-mover's distance error and mean-squared error score for decision tree for different data representations.

We present the branching sequence for the most prominent (i.e., smallest depth) rule for two of the best approaches (the third and the fifth) in Figures 12 and 13. Recall that the X_i inputs are the hand cdf values, with X_1–X_{10} corresponding to player 1's hand probabilities, X_{11}–X_{20} corresponding to player 2's, and then X_{21} corresponding to the card number (Figure 12) or card cdf value (Figure 13). So the third rule for the third representation ($X_{18} <= 0.699$) corresponds to "if the probability that player 2 is dealt at most

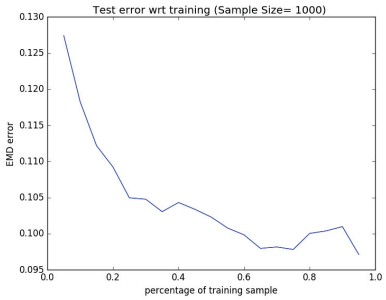

Figure 8: k-nn with $k = 1$ using pdf features.

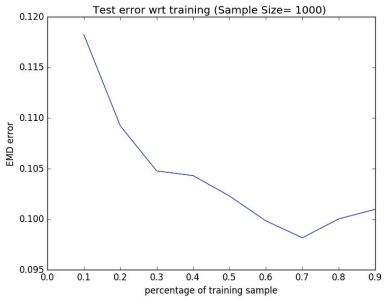

Figure 9: k-nn with $k = 1$ using cdf features.

Figure 10: k-nn with $k = 1$ using pdf features, with one data point for each card (each output is 31 strategy probabilities).

Figure 11: k-nn with $k = 1$ using cdf features, with one data point for each card (each output is 31 strategy probabilities).

an 8 is less than or equal to 0.699 then...” while the first rule ($X_{21} > 6.5$) corresponds to “if player 1’s card is greater than 6.5 then...” Each rule has a branch for true and false values, leading down a path that eventually terminates at a leaf node, which corresponds to a strategy to be played. The leaf node for the given sequence (of all T) in Figure 12 is the strategy that bets 0.1 with probability 1, and for Figure 13 it is the strategy that bets 3.0 with probability 1. While memorizing the entire tree would be infeasible, a human could easily memorize the most important or shallowest-depth rules in advance and apply them during game play.

4.3 Clustering

Inspired by the k-nn results, we decided to explore whether there was a small number of “canonical” instances that were representative of the dataset, so that if a human were to memorize their solutions he could then classify a new instance according to which canonical instance was most similar and implement that strategy. To accomplish this we used the k-medoids clustering algorithm (Kaufman and Rousseeuw 1987), which utilizes an arbitrary distance metric (as opposed to k-means which uses the standard MSE), and we were able to integrate our new generalized earth mover’s distance function. The clustering errors as a function of number of clusters used are reported in Figures 14 and 15. These were based on using 1000 data points, with a training size of 800 and testing size of 200. We find it surprising that the errors were not monotonically decreasing, with the middle value of 10 clusters performing worst for both representations. Overall, using cdf features with only 5 clusters performed best out of all the approaches.

We note that the errors of clustering are somewhat higher than the errors of k-nn and significantly higher than those using decision trees. However, it would be nearly impossible for a human to implement the full decision tree due to its size (though it would be easy to implement several of the most prominent rules). Even k-nn would be extremely difficult for a human to implement, as using 20% of the dataset would have required a human to memorize the full solutions to 200 games in advance, and furthermore to be able to easily determine which of the games a new instance was closest to. Clustering with 5 clusters can be much more easily implemented by a human, since it would only require memorizing solutions to the 5 cluster medoids in advance, as well as an efficient method to determine which of the medoids a new instance was closest to (using the generalized EMD function). This inspires our final approach, which attempts to draw on the advantages of the prior approaches while also precluding the need for a human to perform challenging real-time distance calculations.

4.4 Most interpretable approach: decision tree for cluster medoids

The approaches described previously each have their benefits and limitations, but none produces strategies that can be implemented easily by a human, which was our goal. We now present such an approach. We first perform clustering as described in Section 4.3, and then construct an optimal deci-

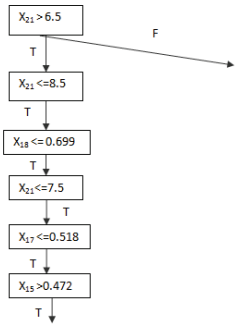

Figure 12: Smallest-depth rule from decision tree with separate datapoints for each hand using cdf features (third representation).

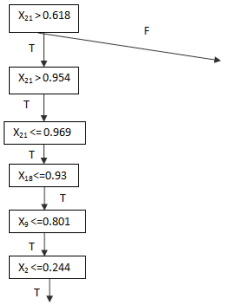

Figure 13: Smallest-depth rule from decision tree with separate datapoints for each hand, using cdf features plus a 21st feature for the cdf of the card (fifth representation).

sion tree from the cluster centers as described in Section 4.2. We used 5 clusters with cdf features, which was shown to perform best. This led to a very simple decision tree: the optimal tree has depth 4 and only 4 rules (Figure 16).

The 5 optimal cluster medoids, as well as the corresponding cdfs of the hand distributions used to generate them, are given in the appendix. One can view these as the main canonical representative game instances. One can see that the strategies and cdf distributions are very different between the medoids. For instance, consider medoid 4. Player 2 is very rarely dealt a 9—less than 1% of the time (the difference between his cdf value of 9 (1) and his cdf value of 8(0.994))—while player 1 is dealt a 9 12.3% of the time. More generally player 1's distribution of strong hands is clearly superior to player 2's (as indicated by the fact that his cdf values for 6, 7, and 8 are significantly lower). The strategy indicates that player 1 chooses a very aggressive action of betting 3.0 with his 9, and betting 0.9 with his 8 for this game. By contrast, for medoid 2 player 2 is dealt a 9 22.3% of the time, and player 1 is forced to take a much more conservative strategy, which only bets 1.3 with a 9 and 0.2 with an 8 (and doesn't even bet at all with a 7).

The most prominent rule in the optimal decision tree is the first branch to the left. Intuitively it says that if the cdf value for player 1 for hand 4 is 0.4273 (i.e., if the probability that player 1 is dealt at most 4 equals 0.4273), then player 1 should output a strategy (the cluster 5 strategy from the appendix). It turns out that this rule matches the training data (consisting of the 5 cluster centers) perfectly in this case (as indicated by the 0.0 value on the top row denoting the EMD error of that depth level). To compute the errors we tested on 1000 data points. The next value is the number of samples considered at each level (e.g., at the second level the sample splits on the left with 1 sample and on the right with 4 samples). The overall EMD error of the decision tree is 0.142, with an accuracy classification score of -1.03. Note that this error is extremely close to that of the clustering, since the decision tree learns the cluster centers almost perfectly. We found it very interesting that we were able to obtain a relatively low error with such a simple tree containing only 4 rules (which only conditions on 3 of the cdf values).

5 Conclusion

We presented a novel formulation of the problem of computing strong game-theoretic strategies that are human understandable as a machine learning problem. Traditionally computing strong strategies in games has fallen under the domain of specialized equilibrium-finding algorithms that produce massive strategy files which are unintelligible to humans. We studied a game that mirrors no-limit Texas hold 'em endgames. Solving these endgames has proven to be a critical component of the strongest poker agents for the most popular variant of poker among humans and a widely recognized AI challenge problem. We proposed a novel formulation where the input features are the private information cdf values and the outputs are the strategy probability vectors, and we devised novel distance functions between pairs of inputs and outputs that generalize the successful earth mover's distance. We also provided a novel procedure for generating random distributions of private information, which we used to create a large database of game solutions. Using the formulation and database, we experimented with several learning algorithms. Experiments with k-nearest neighbors showed that we can achieve low test error by training on only a relatively small percentage of the data. This suggests that there may exist a small set of "canonical" game instances whose solutions could be utilized to obtain good solutions for a variety of unknown instances. Experiments showed that it is feasible to implement decision trees for a large database and to learn simple and useful rules that humans could implement (though it would not be feasible for a human to implement the full strategy indicated by the decision tree). Experiments on k-medoid clustering with our distance metric showed that we can obtain a relatively small error using only a very small number of representative cluster centers. Finally, we presented a new approach that combines the benefits the prior approaches: we first clustered the data into 5 clusters, then constructed a decision tree from these cluster centers. Surprisingly this approach produced relatively small error (as low as clustering and close to that of k-nn). The decision tree is very small and the strategies can be represented compactly, making this approach very easily amenable to human implementation.

We would like to analyze the optimal decision trees to ob-

Figure 14: Clustering error using k-medoids with earth mover's distance with pdf features.

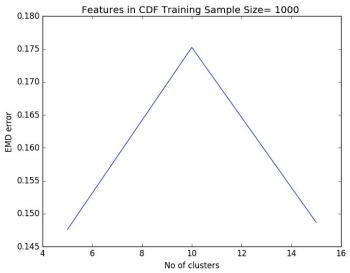

Figure 15: Clustering error using k-medoids with earth mover's distance with cdf features.

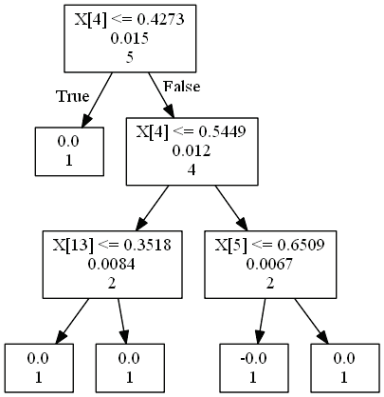

Figure 16: Rules for optimal decision tree from 5 medoids.

tain understandable rules for strong poker strategy, such as the prominent rules we presented. A dual goal of our approach, separate from that of generating strategies for humans to implement, would be to lead to development of improved algorithms for equilibrium computation. It has already been shown that leveraging equilibrium strategy models can improve performance of equilibrium-finding algorithms (Ganzfried and Sandholm 2010). If we are able to compute a small database of solutions for two-player no-limit Texas hold 'em endgames that generalize such that all encountered endgames are close to a game in the database, this could lead to much more efficient endgame solving, since we would not need to do a full equilibrium computation in real time. This could allow endgame solving to be utilized on earlier rounds of the game and not just on the final round. It could also allow endgame solving to be efficiently integrated with the strongest equilibrium-finding algorithm for large imperfect-information games, counterfactual regret minimization (Zinkevich et al. 2007) and its Monte Carlo sampling variants (Lanctot et al. 2009). Rather than have to sample all the way down the tree and store/update regrets and average strategies for the endgames we could look up the endgame strategy for the closest game in the database using our approach. A promising algorithm has been recently devised for integrating endgame solving with offline equilibrium computation using CFR (Burch, Johanson, and Bowling 2014). This approach could be improved by looking up endgame strategies from a presolved database instead of repeatedly solving them during runtime.

There are several avenues we would like to pursue for this project. First, we would like to perform more comprehensive experiments comparing the six data representations we have proposed. We would also like to study theoretical differences between the representations (e.g., characterize exactly the set of game instances that would have identical cdf but different pdf distributions). We would like to evaluate our new strategy distance metrics, which are heuristic and potentially not the best. Note that an effective distance metric between strategies would have many potential applications. For instance, a recent algorithm for opponent exploitation required a procedure to compute the "closest" strategy to a given prior strategy that agreed with the observations, and an approach resembling EMD outperformed L1 and L2 approaches experimentally (Ganzfried and Sandholm 2011). We would also like to implement full-game exploitability as a new metric to evaluate the "cost" of a strategy, which can be integrated with all the learning approaches.

We note that the contributions are not specific to poker games. The model and formulation are general, and would apply to any imperfect-information game where agents are given ordered private information signals. The approaches could also apply to perfect-information games where we can generate a database of games by modifying the values of natural parameters. The approaches are also not specific to two-player zero-sum games, though they do assume that solutions can be computed for the games used in training, which can be more challenging for other game classes.

References

Ankenman, J., and Chen, B. 2006. *The Mathematics of Poker*. ConJelCo LLC.

Bertsimas, D.; Chang, A.; and Rudin, C. 2011. Ordered rules for classification: A discrete optimization approach to associative classification. Operations Research Center Working Paper Series OR 386-11, MIT.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit hold'em poker is solved. *Science* 347(6218):145–149.

Burch, N.; Johanson, M.; and Bowling, M. 2014. Solving imperfect information games using decomposition. In *Pro-*

ceedings of the AAAI Conference on Artificial Intelligence (AAAI).

Ganzfried, S., and Sandholm, T. 2010. Computing equilibria by incorporating qualitative models. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Ganzfried, S., and Sandholm, T. 2011. Game theory-based opponent modeling in large imperfect-information games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Ganzfried, S., and Sandholm, T. 2015. Endgame solving in large imperfect-information games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Ganzfried, S. 2015. Reflections on the first man vs. machine no-limit Texas hold 'em competition. *SIGecom Exchanges* 4.2. To appear in AI Magazine.

Gilpin, A.; Sandholm, T.; and Sørensen, T. B. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Gurobi Optimization, Inc. 2014. Gurobi optimizer reference manual version 6.0.

Johanson, M.; Burch, N.; Valenzano, R.; and Bowling, M. 2013. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Johanson, M. 2013. Measuring the size of large no-limit poker games. Technical report, University of Alberta.

Kaufman, L., and Rousseeuw, P. 1987. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods* 405–416.

Koller, D., and Megiddo, N. 1992. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior* 4(4):528–552.

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte Carlo sampling for regret minimization in extensive games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

Moravcik, M.; Schmid, M.; Ha, K.; Hladik, M.; and Gaukrodger, S. J. 2016. Refining subgames in large imperfect information games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Paruchuri, P.; Pearce, J. P.; Marecki, J.; Tambe, M.; Ordonez, F.; and Kraus, S. 2008. Playing games with security: An efficient exact algorithm for bayesian stackelberg games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Zinkevich, M.; Bowling, M.; Johanson, M.; and Piccione, C. 2007. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

A Distributions and strategies for 5 clusters

Card	P1 cdf	P2 cdf
0	0.1132343	0.0323525
1	0.1547225	0.1672163
2	0.1888979	0.2948809
3	0.3210657	0.3365564
4	0.4975004	0.6160222
5	0.623888	0.6887151
6	0.7403509	0.7403384
7	0.857168	0.8101873
8	0.8853638	0.8697037
9	1.0	1.0

Table 2: Medoid 1 hand distributions.

Card(s)	Bets(probabilities)
0	1.3(1)
1	0.1(0.225), 0.3(0.088), 1.2(0.292), 1.3(0.395)
2–4	0(1)
5–7	0.1(1)
8	1.2(1)
9	1.2(0.252), 1.3(0.748)

Table 3: Medoid 1 strategy.

Card	P1 cdf	P2 cdf
0	0.1965438	0.0843366
1	0.2133783	0.129227
2	0.3845294	0.2263237
3	0.5221381	0.2880686
4	0.5768681	0.3534791
5	0.6698331	0.5186297
6	0.6893295	0.6870242
7	0.8345081	0.7378451
8	0.9574423	0.7769845
9	1.0	1.0

Table 4: Medoid 2 hand distributions.

Card(s)	Bets(probabilities)
0	0(0.639), 0.2(0.021), 1.3(0.340)
1–7	0(1)
8	0.2(1)
9	1.3(1)

Table 5: Medoid 2 strategy.

Card	P1 cdf	P2 cdf
0	0.0270415	0.1397059
1	0.1150136	0.3303171
2	0.2877046	0.33761
3	0.4277975	0.5392525
4	0.6064050	0.5960327
5	0.6319442	0.7613663
6	0.7270908	0.8692346
7	0.8002724	0.8889503
8	0.8425349	0.9618729
9	1.0	1.0

Table 6: Medoid 3 hand distributions.

Card(s)	Bets(probabilities)
0	0(0.591), 0.1(0.002), 0.3(0.010), 0.6(0.362), 2.0(0.035)
1–5	0(1)
6	0.1(1)
7	0.3(1)
8	0.6(1)
9	2.0(1)

Table 7: Medoid 3 strategy.

Card	P1 cdf	P2 cdf
0	0.1516277	0.1718702
1	0.2031077	0.2505481
2	0.3500789	0.2615690
3	0.3909628	0.3670855
4	0.5128757	0.5872985
5	0.5226815	0.6486966
6	0.6294331	0.6984665
7	0.7380967	0.8388290
8	0.8868218	0.9935035
9	1.0	1.0

Table 8: Medoid 4 hand distributions.

Card(s)	Bets(probabilities)
0	0.1(0.003), 0.9(0.647), 3.0(0.350)
1	0.9(1.0)
2	0(0.981), 0.1(0.019)
3–6	0(1)
7	0.1(1)
8	0.9(1)
9	3.0(1)

Table 9: Medoid 4 strategy.

Card	P1 cdf	P2 cdf
0	0.0066483	0.0668309
1	0.0588898	0.1650387
2	0.2279267	0.2450563
3	0.3064846	0.3245739
4	0.3571740	0.4779984
5	0.5144995	0.5027299
6	0.6154961	0.7156640
7	0.7699768	0.8044380
8	0.9093516	0.8223335
9	1.0	1.0

Table 10: Medoid 5 hand distributions.

Card(s)	Bets(probabilities)
0	0.4(1.0)
1	1.0(1.0)
2	0.1(1.0)
3	0.1(0.068), 0.4(0.530), 1.0(0.370), 2.8(0.033)
4	0 (0.915), 0.1(0.085)
5	0(1)
6	0.1(1)
7	0.4(1)
8	1.0(1)
9	2.8(1)

Table 11: Medoid 5 strategy.