

Learning Strategies for Opponent Modeling in Poker

Ömer Ekmekci

Department of Computer Engineering
Middle East Technical University
06800 Ankara, Turkey
oekmekci@ceng.metu.edu.tr

Volkan Şirin

Department of Computer Engineering
Middle East Technical University
06800 Ankara, Turkey
volkan.sirin@ceng.metu.edu.tr

Abstract

In poker, players tend to play sub-optimally due to the uncertainty in the game. Payoffs can be maximized by exploiting these sub-optimal tendencies. One way of realizing this is to acquire the opponent strategy by recognizing the key patterns in its style of play. Existing studies on opponent modeling in poker aim at predicting opponent's future actions or estimating opponent's hand. In this study, we propose a machine learning method for acquiring the opponent's behavior for the purpose of predicting opponent's future actions. We derived a number of features to be used in modeling opponent's strategy. Then, an ensemble learning method is proposed for generalizing the model. The proposed approach is tested on a set of test scenarios and shown to be effective.

Introduction

Poker is a game of imperfect information in which players have only partial knowledge about the current state of the game (Johanson 2007). It has many variants where a widely popular one is Texas Hold'em which is considered to be the most strategically complex poker variant (Billings et al. 1998). Features like having very large state spaces, containing stochastic outcomes, risk management, multiple competitive agents, agent modeling, deception and handling unreliable information also constitute the fundamentals of AI and multiagent systems (Billings et al. 1998).

One of the important features separating poker from the other games is opponent modeling. In games of perfect information, not regarding whether your opponent plays optimally or not, playing optimally will suffice. On the contrary, this strategy will not work in poker. Against sub-optimal opponents, maximizing player will gain much higher payoffs than an optimal player, since optimal players make decisions without regarding the context, whereas maximizing players exploit opponent's weaknesses and adjust their play accordingly. For example, against excessively bluffing opponents, it would not be correct to fold with hands having high winning probability however against tight opponents folding may be a proper action. Thus exploiting the sub-optimal tendencies of opponents or predicting how much they deviate from the equilibrium strategy with what type of hands,

will dramatically increase the payoffs. In conclusion, opponent modeling is very essential for building a world-class poker player.

Having stated that opponent modeling is imperative, two questions arose for that matter which their solutions are not quite straightforward. First problem is determining what statistics of an opponent should be generated, i.e. extracting features, for accurate prediction of hands and/or future actions. The other problem is how to use these predictions to exploit the opponent for making better decisions. While in this study we try to address a solution to the first question using machine learning techniques, the second question is an equally complex problem for opponent modeling in poker.

In this paper, we propose an opponent modeling system for Texas Hold'em poker. First, this paper explains research done on opponent modeling. In the second part, rules of Texas Hold'em poker are introduced. Then, the statistical elements for opponent modeling in Texas Hold'em poker game is presented. Important features are analyzed in detail to be used in machine learning algorithms. Finally, we render the main work of this study. We built up a machine learning system for the opponent modeling problem. The experiments are presented in detail and the results are discussed.

For the experiments, the hand histories of 8 players (among 19 players) participating in AAAI Annual Computer Poker Competition (ACPC) 2011 are chosen, which are Calamari, Sartre, Hyperborean, Feste, Slumbot, ZBot, 2Bot and LittleRock (in the descending order of their performances in the competition).

Related Work

Even though opponent modeling is a key feature to play poker well (Billings et al. 2002) and research on computer poker sped up about a decade ago, still research on opponent modeling remains sparse.

(Billings et al. 1998) proposed an agent with opponent modeling, named as *Loki*, for the limit variant of the Texas Hold'em poker. For opponent modeling purposes, many factors influencing a player's style of playing are introduced. These factors constitute and alter (as new actions are observed) probabilities of the possible hands assigned to the opponent. Nevertheless, the opponent model developed is somewhat a simplistic model that more sophisticated fea-

tures of opponent behavior need to be considered to build a more powerful model.

Poki, a new improved version of the *Loki*, is proposed by Davidson et al., which uses Neural Networks to assign each game state a probability triple (Davidson et al. 2000). Many features such as number of active players, relative betting position, size of the pot, and characteristics of the community board cards are used to represent the game state. They reported a typical prediction accuracy around %80.

There are numerous important game theory based studies for opponent modeling. One promising research is done by (Billings et al. 2006), proposing a new agent named *VexBot* which is based on adaptive game-tree algorithms: *Maximax* and *Miximax*. These algorithms compute the expected value of each possible action which needs opponent statistics. For this purpose, an opponent model is built by computing the rank histogram of opponents hand for different betting sequences. Hands are categorized as one of a broader hand groups according to their strength. Since this approach is strictly based on opponent specific criteria, modeling the opponent accurately is obligatory. Unfortunately, they did not give a performance metric for their opponent modeling module, which is part of a larger system. A newer work proposed an efficient real-time algorithm for opponent modeling and exploitation based on game theory (Ganzfried and Sandholm 2011). Rather than relying on opponent specific hand history or expert prior distributions, their algorithm builds an opponent model by observing how much the opponent deviates from the equilibrium point by means of action frequencies. One other powerful side is that the algorithm can exploit an opponent after several interactions. Nevertheless, we agree with the author that the algorithm has a crucial weakness that it makes the agent highly exploitable against strong opponents. In contrast, off-game modeling of the opponent, with hand histories or other opponent specific data, does not have this weakness.

Finally, Bayesian approaches are used for opponent modeling in poker. An earlier study on this subject (Korb, Nicholson, and Jitnah 1999), focused on Five-Card Stud poker and tried to classify the opponent's hand into some predefined classes using the information available for the purpose of estimating the probability of winning the game. Whereas a more recent study (Southey et al. 2005), focusing on two hold'em variants, generated a posterior distribution over opponent's strategy space which defines the possible actions for that opponent. Then having built the opponent model, different algorithms for computing a response are compared. Another important study (Ponsen et al. 2008), proposed an opponent model, predicting both outcomes and actions of players for No-Limit variant of Texas Hold'em. This model relies heavily on opponent specific data which shapes the corrective function for modeling the behavior of opponent by altering the prior distribution.

Texas Hold'em

In this study heads-up and limit variant of Texas Hold'em poker is focused, in which there are only two players. Each player has 2 cards named as *hole cards*. There are four phases of a game: *pre-flop*, *flop*, *turn* and *river*. Last three

of these called the post-flop phases. Three possible actions are allowed in the game: *fold*, *call* and *raise*. Fold means the player withdraws from the game and forfeits previously committed money. Call (or check) means the player commits the same amount of the money with his opponent and the phase ends. Raise (or bet) means the player commits more money than his opponent. Up to four raises are allowed in each round.

In the *pre-flop* phase, first player (small blind) contributes 1 unit of money and the other player, which is also the dealer, contributes 2 units of money automatically. Then, each player is dealt two hidden (hole) cards. Then a betting round begins until a player folds or calls. In the *flop* phase, three community cards are dealt to the table. Every player can see and use community cards. Another similar betting round begins in this phase, except the blind bets. In each of the *turn* and *river* phases, one more community card is dealt and a new betting round is conducted.

Opponent Modeling

There are two crucial aspects of opponent modeling in poker observed in the literature. One is categorizing opponent's hand into one of predefined broader hand groups or generating a probability distribution for the possible hands which can presumably be altered as game proceeds, while the other one is predicting opponent's actions based on the information available in the game. In this particular study, we will focus on how to generate a machine learning system for predicting the opponent's action for a given situation.

As mentioned in the preceding section, there are four phases and three potential actions that a player can take in the game. In each phase, there may be zero or more *decision points*, i.e. just the moment before the opponent makes its move. For example; if players folded in flop then since the game ends, there will be no turn or river phase hence no decision points. We approached this decision problem as a classification problem, where the labels correspond to one of the actions and the feature vector is composed of several features being extracted and containing information from the current state of the game. Next, the most crucial elements for opponent modeling namely, features, are explained.

Feature Analysis and Selection

Features are the core elements of an opponent modeler designed by machine learning approaches. They are the basic building blocks for determining how a poker player plays. In order to develop a modeler for accurately predicting opponents' future actions, statistics and certain measures which are used as features, should be collected and analyzed very carefully. A modeler with redundant features may lead to high computational complexity and low performance whereas missing features may lead to inaccurate predictions. Moreover, since playing styles of players vary vastly, the subset of features which models a player accurately, may not apply to another player.

There are several factors that affect a decision of a player. The obvious elements are the hole cards and community cards. There are $C(52, 2) = 1326$ possibilities for hole cards

Table 1: Candidate features

Id	Explanation
1	Hand strength
2	PPot
3	NPot
4	Whether the player is dealer or not.
5	Last action of the opponent (null, call or raise)
6	Last action of the opponent in context (null, check, call, bet or raise)
7	Stack (money) committed by the player in this phase
8	Stack committed by the opponent in this phase
9	Number of raises by the the player in this phase
10	Number of raises by the opponent in this phase
11	Hand rank
12	Winning probability
13	Hand outs
14	Number of raises by the player in previous phases
15	Number of raises by the opponent in previous phases
16	Highest valued card on the board
17	Number of queens on the board
18	Number of kings on the board
19	Number of aces on the board

and there are $C(52, 5) = 2598960$ board configurations. Exposing card sets to a machine learner as a feature is subject to fail because of this very high dimensionality. Therefore, researchers usually convert card information to a couple of measures using various algorithms called *hand evaluation algorithms* (Billings 2006). Hand evaluation algorithms aim to assess the strength of the hand of the agent. There are different approaches to evaluate the hand in the pre-flop phase and the post-flop phases.

For the pre-flop phase, the winning probabilities of each hand can be approximated by using a simple technique that is called roll-out simulation (Billings 2006). The simulation consists of playing several million trials. In each of these trials, after hidden cards are dealt, all other cards are dealt without any betting and the winning pair is determined. Each trial in which a pair wins the game, increases the value of that particular hand. It is an oversimplification of the game, however it provides an accurate description of relative strengths of the hands at the beginning of the game.

For the post-flop phases, there are two algorithms for hand assessment, *hand strength* and *hand potential* (Billings 2006). The hand strength, *HS*, is the probability of a given hand is better than the possible hands of active opponent. To calculate HS, all of the possible opponent hands are enumerated and checked whether our agent’s hand is better, tied or worse. Summing up all of the results and dividing the number of possible opponent hands give the hand strength. Hand potential calculations are for calculating the winning probability of a hand when all the board cards are dealt. The first time the board cards are dealt, there are two more cards to be revealed for each round. For the hand potential calculation, we look at the potential impact of these cards. The positive potential, *PPot*, is the probability of increase of a hand’s rank after the board cards are dealt. The negative potential, *NPot*, is the count of hands which will make a leading hand end up behind. PPot and NPot are calculated by enumerating all the

possible cards for the opponent like the enumeration in hand strength, in addition by enumerating all the possible board cards to be revealed.

There are several measures that can assess the hand from other perspectives. Two of these measures are *hand rank* and *hand outs*. Hand rank refers to the relative ranking of a particular five card hand by the rules of the game at showdown i.e. the player with highest rank wins the game. Hand outs refers the number of possible cards that may be opened to the table and improve the rank of the hand. Finally, *the winning probability* (at showdown), can be computed based on hand strength and hand potential as in (Felix and Reis 2008):

$$P(win) = HS \times (1 - NPot) + (1 - HS) \times PPot \quad (1)$$

Not only the card measures but also the table context considerably affects the decision of a player, few examples are committed portion of stack, number of raises. Complete set of features that are used for our analysis are given in Table 1. It is important to note that a number of features are not applicable for river and pre-flop phases. For example, hand potential is not meaningful for the river phase because all the board cards are already dealt. Number of features may seem to be small, but note that first three features embody lots of information about the present and future of the game. Moreover, players’ style may vary according to current phase being conducted. However, since feature subset varies in diverse phases, this factor is processed differently which is explained later.

Before the classification step, an optimal subset of candidate features has to be selected that leads to the greatest performance according to a predefined optimality criteria. For this purpose, a feature selection algorithm must be employed. In our problem, feature selection is the process of selecting a particular subset of features for each player and game phase.

As explained in (Liu and Yu 2005), a feature selection algorithm consists of four parts: Subset Generation, Subset Evaluation, Stopping Criterion and Result Validation. Backward elimination and forward selection are the basic subset generation algorithms which find the optimal feature set iteratively.

Backward elimination technique performs the iterations starting with full feature set. In each iteration all possible N-1 subsets of the N-sized feature set are generated. Then the least useful feature whose absence improves the performance most is removed. Whereas, forward selection procedure starts with empty set and adds new features progressively to the current set. As explained in (Guyon and Elisseeff 2003), backward elimination has a certain advantage over forward selection method. A feature may be more useful in the presence of another certain feature so if the algorithm started with a small feature set, a good combination might be missed. Hence, backward elimination can give better results. However, it can also be computationally more expensive because it works on larger sets than the forward selection algorithm.

In our experiments, sequential backward elimination and forward election techniques are performed and results are

Table 2: Steps during backward elimination for the data of LittleRock flop phase, F represents the whole feature set

Feature Set	Validation Accuracy
F	88.2%
$F - \{16\}$	88.6%
$F - \{1, 16\}$	88.8%
$F - \{1, 7, 16\}$	88.8%
$F - \{1, 5, 7, 16\}$	88.8%
$F - \{1, 5, 6, 7, 16\}$	88.8%
$F - \{1, 5, 6, 7, 9, 16\}$	88.8%
$F - \{1, 5, 6, 7, 9, 10, 16\}$	88.8%
$F - \{1, 5, 6, 7, 9, 10, 11, 16\}$	88.8%

Table 3: Steps during forward selection for the data of LittleRock flop phase.

Feature Set	Validation Accuracy
$\{14\}$	61.5%
$\{6, 14\}$	77.5%
$\{6, 12, 14\}$	83.2%
$\{1, 6, 12, 14\}$	86.4%
$\{1, 6, 12, 14, 16\}$	87.3%
$\{1, 6, 12, 14, 15, 16\}$	87.9%
$\{1, 2, 6, 12, 14, 15, 16\}$	88.1%
$\{1, 2, 6, 12, 13, 14, 15, 16\}$	88.3%
$\{1, 2, 6, 9, 12, 13, 14, 15, 16\}$	88.4%
$\{1, 2, 3, 6, 9, 12, 13, 14, 15, 16\}$	88.5%
$\{1, 2, 3, 4, 6, 9, 12, 13, 14, 15, 16\}$	88.5%

compared. As for the evaluation criteria for each subset, cross-validation accuracy is used. Moreover, the algorithms are terminated when addition or subtraction of a feature cannot generate better performance than the previous iteration. Table 2 shows the steps during reduction for the player LittleRock in flop phase of the game. First two exclusions increase the performance significantly but the later ones are insignificant. After ninth step, a better subset cannot be found and hence the algorithm terminates. Feature sets, produced by two methods, are similar but not exactly the same. Features with id 1, 6, 9 and 16 are excluded by backward reduction but included in forward selection. In comparison, backward elimination gives slightly better performance throughout the entire data set.

Table 3 shows the steps for the player LittleRock in flop phase using forward selection. Expansion stops after eleventh step since a subset with 12 features cannot be found better than the current 11-sized subset.

Even though backward elimination method leads to better results than the forward selection algorithm, it is computationally expensive since it processes each $N-1$ subset of N -sized feature set at each step. It would be even much more time consuming to perform this analysis if the number of opponents to be modeled increases. Therefore, an algorithm that can provide the relative importance of the features is needed. If such information were available, instead of trying each subset, the least important one could have been dropped. Relief-F is a popular feature selection algorithm for obtaining that information.

Relief-F is an algorithm introduced in (Kononenko 1994),

Table 4: Steps during reduction of feature set guided with Relief-F

Feature Set	Validation Accuracy
F	88.2%
$F - \{7\}$	82.0%
$F - \{7, 9\}$	82.4%
$F - \{7, 9, 17\}$	82.7%
$F - \{7, 9, 17, 18\}$	82.7%
$F - \{7, 9, 11, 17, 18\}$	82.6%
$F - \{7, 9, 11, 17, 18, 19\}$	82.5%
$F - \{2, 7, 9, 11, 17, 18, 19\}$	77.6%
$F - \{2, 7, 9, 11, 13, 17, 18, 19\}$	77.4%
$F - \{2, 7, 8, 9, 11, 13, 17, 18, 19\}$	76.3%
$\{1, 4, 5, 6, 10, 12, 14, 15, 16\}$	76.2%
$\{1, 4, 5, 6, 10, 12, 14, 15\}$	76.2%
$\{1, 5, 6, 10, 12, 14, 15\}$	75.9%
$\{1, 5, 6, 10, 12, 14\}$	75.0%
$\{1, 5, 6, 12, 14\}$	70.0%
$\{1, 5, 6, 14\}$	69.3%
$\{5, 6, 14\}$	68.3%
$\{5, 14\}$	52.0%
$\{14\}$	49.5%

Table 5: Selected feature sets for different players

Player	Feature Set
2Bot	$\{2, 3, 4, 9, 10, 11, 12, 13, 14, 15\}$
LittleRock	$\{2, 3, 4, 8, 12, 13, 14, 15, 17, 18, 19\}$
Slumbot	$\{1, 2, 3, 4, 8, 11, 12, 14, 15\}$
ZBot	$\{1, 3, 5, 8, 12, 13, 14, 15, 16, 18\}$
Feste	$\{1, 3, 4, 8, 11, 12, 13, 14, 15\}$
Hyperborean	$\{2, 3, 6, 10, 12, 13, 14, 16\}$
Sartre	$\{2, 6, 8, 12, 13, 14, 15\}$
Calamari	$\{1, 2, 3, 4, 6, 9, 12, 13, 14, 15, 16\}$

which is a multiclass extension to the original Relief algorithm by (Kira and Rendell 1992). It is used for estimating the quality of individual features with respect to the classification.

After performed Relief-F and getting weights, the features are sorted in ascending order. Then, at each iteration, the feature with the lowest weight is dropped. Finally, the feature set with maximum validation accuracy is selected. As seen in Table 4, dropping the features suggested by Relief-F did not improve performance for the entire data set. Hence, feature subsets generated by backward reduction algorithm used integrated in the following machine learning systems rather than Relief-F algorithm.

In Table 5, selected feature sets for each player produced by the backward selection algorithm is presented. They are similar but not exactly the same which is expected since the style of each player varies as stated in the earlier sections.

Learning an Opponent Model

Then next step after feature selection procedure is classification. For mapping the features to decisions, i.e. discovering the complex relations between them, we employed three machine learning algorithms: *Neural Networks* (Bishop 1995), *Support Vector Machines* (Cortes and Vapnik 1995) and *K*

Table 6: Test accuracies of classifiers of all players for all phases

	Pre-Flop			Flop			Turn			River		
	NN	SVM	KNN	NN	SVM	KNN	NN	SVM	KNN	NN	SVM	KNN
Sartre	90%	90%	100%	89%	96%	97%	94%	96%	96%	90%	94%	95%
Hyperborean	87%	88%	96%	84%	85%	84%	81%	83%	82%	82%	83%	83%
Feste	89%	90%	96%	87%	87%	89%	82%	82%	84%	81%	84%	86%
ZBot	79%	80%	93%	76%	81%	80%	79%	80%	79%	82%	83%	82%
Slumbot	88%	88%	95%	80%	80%	79%	79%	81%	78%	77%	77%	76%
LittleRock	87%	86%	98%	87%	87%	89%	81%	83%	83%	81%	83%	86%
2Bot	92%	92%	95%	85%	84%	86%	78%	78%	78%	77%	78%	78%
Calamari	88%	89%	99%	95%	87%	97%	93%	85%	94%	93%	84%	94%

Nearest Neighbors (Wettschereck, Aha, and Mohri 1997).

For each of the mentioned player in the very first section, data gathered contains around 30 thousand games. In each game, there are at most four phases. Each of the decision points in the game are treated as a sample. As a result, typical size of the data for a player is around 30 thousand samples for pre-flop phase, 20 thousands for flop, 10 thousands for turn and 5 thousands for river. The data is divided into two distinct parts: training and test, which test set contains 20% of the samples in the whole data.

After feature selection, experiments start with the training phase which includes cross-validation for parameter optimization where *K-fold Cross-Validation* is used (Kohavi 1995). Next, the best parameter set should be determined. Using *Grid Search* (Hsu, Chang, and Lin 2003), for each parameter, we define a set of values to test. In grid search procedure, we exhaustively compute the validation accuracy for all possible parameter sets and take the one giving maximum score. After finishing selecting the parameters for the three classifiers, their performances are determined using the test set.

First, for neural networks, 3-layer structure is used. The network has 3 output nodes representing the unnormalized probabilities of fold, call and raise actions. The decision is the action with the highest value and the activation function for the nodes is the sigmoid function. The weights are learned using the method of *gradient descent with momentum* where the error function is sum of squares. Hence in parameter optimization epoch, number of hidden nodes, learning rate (η), momentum (α) are used.

Number of hidden layer nodes is searched around the number of selected features. We observed that neural network performance is generally stable around this number. Best cross-validation performances obtained when η is around 0.1 and epoch number is around 4000, where η is validated for values in the range [0.1,0.5] and epoch number for various values between 100 and 10000. Moreover, we also observed that introducing momentum to the neural network training process does not improve performance and in general it is not sensitive to the varying parameters for this data. Results for all the players can be seen from Table 6.

Second, for SVM, we investigated the effect of varying the C parameter. For most of the players validation accuracies are stable after $C \geq 4$ (up to 64 increasing with multiples of 2), and best is obtained when C is around 8. Linear and radial basis function has been used as kernel. For the

RBF kernel, in addition to C , γ parameter is optimized in the interval [0.25, 4] in which the best performance is observed for $\gamma = 2$. In the experiments, SVM produced about 4-5% better results with RBF kernel than with linear kernel. Table 6 shows testing accuracies for different players for SVM with RBF kernel only.

Finally, before applying KNN algorithm, data is normalized. If the range of a feature is significantly larger than the others, this feature can dominate the distance calculation. For that reason, all features are mapped to [0, 1] range so that each of them contributes equally to the Euclidean distance metric. We investigated the effect of varying the k parameter with 10-fold cross validation. Generally, validation accuracy increased monotonically from $k = 2$ to about $k = 8$ which after no significant change is observed up to $k = 20$, and best validation accuracy is obtained about $k = 12$. Table 6 shows the test results for all the players.

First, the results are really promising, machine learning approaches proved themselves to be very successful. Second, in pre-flop and flop phases, in which more data is available, classifiers have better performance than the others. Moreover, KNN performed slightly better than the others where neural network gave an average accuracy of around $85\% \pm 3\%$, SVM $86\% \pm 3\%$ and KNN $91\% \pm 6\%$ for all phases. Finally, an interesting pattern stands out in Table 6. According to total bankroll results in the competition, Calamari ranked as first and Sartre as second, the most successful bots in the competition turn out to be most predictable by our classifiers.

Learning Different Styles and Generalizing The Model

Until now, the learning system is trained and tested with the same data. What would happen if we tested the suggested learning system with a different scenario? In fact, this is a very common situation with human players. They observe some players and later they try to benefit from their previous observation with other players. When the models, generated with neural networks trained with the data of a particular player, are tested with another player's data, performances generally drop. For example; when the test data of Sartre is presented to a model for Calamari, the performance becomes 83%, however it was 90% when tested with its own model, or performance for Calamari drops from 93% to 81% when it is presented to model of Sartre rather than

Table 7: Ensemble accuracy compared with experts

	Testing Against						Avg.
	Hyperborean	Calamari	Sartre	LittleRock	Slumbot	2Bot	
Hyperborean	81%	90%	80%	79%	71%	75%	79%
Calamari	80%	93%	83%	81%	73%	75%	81%
Sartre	73%	81%	90%	75%	71%	71%	77%
LittleRock	80%	90%	86%	82%	75%	75%	81%
Ensemble	82%	92%	89%	82%	76%	77%	83%

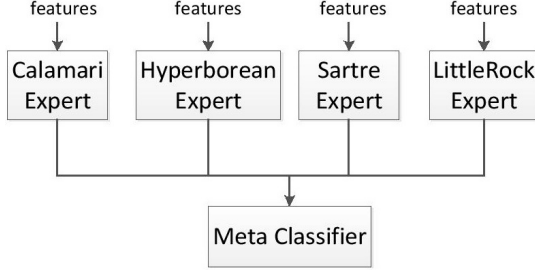


Figure 1: Stacked Generalization for Opponent Modeling

itself. This can be a problem for an artificial agent when it confronts a particular opponent for the first time having no history with that opponent. Even though some data is available about that opponent the feature selection procedure can be time-consuming, hence not feasible. To overcome these problems, we proposed an ensemble system that generalized the system eliminating feature analysis and predict even more accurately the decision of opponents.

Ensemble Learning

Ensemble learning refers to the idea of combining multiple classifiers into a more powerful system with a better prediction performance than the individual classifiers (Rokach 2010). Alpaydin covers seven methods of ensemble learning: voting, error-correcting output codes, bagging, boosting, mixtures of experts, stacked generalization and cascading in his book (Alpaydin 2010).

In our case, several experts are developed, each trained on a particular player. We propose an ensemble scheme of stacked generalization which best fits to the problem. To overcome this problem, we set up the system which is illustrated in Figure 1. Having performed some randomized selections for experts from all of the players and cross-validate, we select the following: Calamari, Sartre, Hyperborean and LittleRock. In this system, the output of each expert learner is fed to the meta classifier as a feature, which are all neural networks having 3 output nodes. Then they are concatenated to make a feature vector with 12 entries, fed to the meta-classifier, which is also a neural network with 3 output nodes.

For this experiment, data is split into 3 parts. First part is used to train experts. For these experts, as presented before, features are selected individually for the purpose of best representing their "expertise" and models are trained with their corresponding data. Second part is used to train the meta

classifier. Data of four different players are fed into each expert and their predictions are obtained. After that, their outcomes are concatenated and fed into the meta-classifier as input. Then, a regular supervised training is performed to the meta-classifier. Finally, third part is used to test the ensemble system. Table 7 shows the results for this system.

In Table 7, we see that when testing against a certain player, the expert of that player gives the best performance. This can be seen from the first four diagonal entries. Ensemble usually comes the second after the expert when predicting a particular player. We see that ensemble has managed to model different players with fairly stable performance. The average performance of the ensemble is greater than the individual experts. In Table 7, it is observed that the stacked generalization model is able to predict the actions of the new opponents better than the experts i.e. the models generated with the traditional machine learning classifiers.

Conclusion and Future Work

In this study, a new opponent modeling system for predicting the behaviour of opponents in poker by using machine learning techniques is proposed. We observe that machine learning approaches are found to be very successful for inferring the strategy of opponents. This is achieved by careful selection of features and parameters for each of the individual players. By using the selected feature set, we carried out a number of experiments to demonstrate the effectiveness, and found that KNN algorithm has the best performance. For the purposes of generalizing the model and alleviating the computational burden of feature selection, an ensemble model is introduced. Rather than performing feature selection for all of the newly confronted players, it will be performed for only the experts. In a stacked generalization scheme, the obtained results are promising. However, experiments for ensemble learning can be performed using different classifiers, where only neural network is presented in this study.

After successfully building the models for opponents, the next major step is to determine how to use these results for a betting strategy. It is important to note that the opponent models can also be used as a betting model for mimicking the modeled opponent. The predictions can be interpreted as the actions to be taken i.e. decisions of the agent. Moreover, the predictions of opponent modelers can be synthesized with game-theoretic approaches for betting mechanisms. For this purpose fuzzy classifiers can also be beneficial in order to get the prediction probabilities of actions to be used in betting algorithms.

References

- Alpaydin, E. 2010. *Introduction to Machine Learning*. The MIT Press.
- Billings, D.; Papp, D.; Schaeffer, J.; and Szafron, D. 1998. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence*, 493–499. John Wiley & Sons LTD.
- Billings, D.; Davidson, A.; Schaeffer, J.; and Szafron, D. 2002. The challenge of poker. *Artificial Intelligence* 134(1):201–240.
- Billings, D.; Davidson, A.; Schauenberg, T.; Burch, N.; Bowling, M.; Holte, R.; Schaeffer, J.; and Szafron, D. 2006. Game-tree search with adaptation in stochastic imperfect-information games. *Computers and Games* 21–34.
- Billings, D. 2006. *Algorithms and assessment in computer poker*. Ph.D. Dissertation, University of Alberta.
- Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc.
- Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.
- Davidson, A.; Billings, D.; Schaeffer, J.; and Szafron, D. 2000. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI’00*, 1467–1473.
- Felix, D., and Reis, L. 2008. An experimental approach to online opponent modeling in texas hold’em poker. *Advances in Artificial Intelligence-SBIA 2008* 83–92.
- Ganzfried, S., and Sandholm, T. 2011. Game theory-based opponent modeling in large imperfect-information games. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.
- Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *The Journal of Machine Learning Research* 3:1157–1182.
- Hsu, C.; Chang, C.; and Lin, C. 2003. *A Practical Guide to Support Vector Classification*. Technical Report, Department of Computer Science, National Taiwan University.
- Johanson, M. 2007. Robust strategies and counter-strategies: Building a champion level computer poker player. In *Masters Abstracts International*, volume 46.
- Kira, K., and Rendell, L. 1992. A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine learning*, 249–256. Morgan Kaufmann Publishers Inc.
- Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, volume 14, 1137–1145.
- Kononenko, I. 1994. Estimating attributes: analysis and extensions of relief. In *Machine Learning: ECML-94*, 171–182. Springer.
- Korb, K.; Nicholson, A.; and Jitnah, N. 1999. Bayesian poker. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 343–350. Morgan Kaufmann Publishers Inc.
- Liu, H., and Yu, L. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering* 17(4):491–502.
- Ponsen, M.; Ramon, J.; Croonenborghs, T.; Driessens, K.; and Tuyls, K. 2008. Bayes-relational learning of opponent models from incomplete information in no-limit poker. In *Twenty-third Conference of the Association for the Advancement of Artificial Intelligence (AAAI-08)*.
- Rokach, L. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33(1):1–39.
- Southey, F.; Bowling, M.; Larson, B.; Piccione, C.; Burch, N.; Billings, D.; and Rayner, C. 2005. Bayes bluff: Opponent modelling in poker. In *In Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence*.
- Wettschereck, D.; Aha, D.; and Mohri, T. 1997. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review* 11(1):273–314.